**SANDIA REPORT**
SAND2018-4545
Unlimited Release
Printed April, 2018

# Artificial Diversity and Defense Security (ADDSec) Final Report

Adrian R Chavez, Jason R Hamlet, and William MS Stout

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories

SAND2018-4545
Printed April 2018
Unlimited Release

# Artificial Diversity and Defense Security (ADDSec) Final Report

Adrian R Chavez, Jason R Hamlet, and William MS Stout
Networked Systems Survivability & Assurance, Assurance Technologies and Assessments,
Systems Security Research, Computer Systems Security Analysis R&D,
and Cyber Security Initiatives
Sandia National Laboratories
P. O. Box 5800
Albuquerque, New Mexico 87185-MS0672

## Abstract

Critical infrastructure systems continue to foster predictable communication patterns and static configurations over extended periods of time. The static nature of these systems eases the process of gathering reconnaissance information that can be used to design, develop, and launch attacks by adversaries. In this research effort, the early phases of an attack vector will be disrupted by randomizing application port numbers, IP addresses, and communication paths dynamically through the use of overlay networks within Industrial Control Systems (ICS). These protective measures convert static systems into "moving targets," adding an additional layer of defense. Additionally, we have developed a framework that automatically detects and defends against threats within these systems using an ensemble of machine learning algorithms that classify and categorize abnormal behavior. Our proof-of-concept has been demonstrated within a representative ICS environment. Performance metrics of our proof-of-concept have been captured with latency impacts of less than a millisecond, on average.

## ACKNOWLEDGMENTS

The authors acknowledge the work that produced the results presented in this paper was funded by the U.S. Department of Energy/Office of Electricity Delivery and Energy Reliability (DOE/OE) as part of the Cybersecurity for Energy Delivery Systems (CEDS) Program.

## TABLE OF CONTENTS

## FIGURES

## TABLES

## NOMENCLATURE

| Abbreviation | Definition |
|---|---|
| **API** | Application Programming Interface |
| **DDoS** | Distributed Denial of Service |
| **DHCP** | Dynamic Host Configuration Protocol |
| **DoS** | Denial of Service |
| **FN** | False Negative |
| **FP** | False Positive |
| **ICMP** | Internet Control Message Protocol |
| **ICS** | Industrial Control System |
| **IDS** | Intrusion Detection System |
| **IED** | Intelligent Electronic Device |
| **IoT** | Internet of Things |
| **IP** | Internet Protocol |
| **IPv4** | IP Version 4 |
| **IT** | Information Technology |
| **MAC** | Media Access Control |
| **MCC** | Matthew's Correlation Coefficient |
| **MTD** | Moving Target Defense |
| **NAT** | Network Address Translation |
| **NTP** | Network Time Protocol |
| **PLC** | Programmable Logic Controller |
| **RTU** | Remote Telemetry Unit |
| **SDN** | Software Defined Networking |
| **SOCKS** | Socket Secure |
| **SPAN** | Switched Port Analyzer |
| **SVM** | Support Vector Machine |
| **TCP** | Transmission Control Protocol |
| **TN** | True Negative |
| **TP** | True Positive |
| **UDP** | User Datagram Packet |
| **VPN** | Virtual Private Networks |

# 1. INTRODUCTION

Historically, control systems have primarily depended upon their isolation [1] from the Internet and from traditional Information Technology (IT) networks as a means of maintaining secure operation in the face of potential remote attacks over computer networks. However, these networks are incrementally being upgraded [2] and are becoming more interconnected with external networks so they can be effectively managed and configured remotely. Examples of control systems include the electric power grid, smart grid networks, micro grid networks, oil and natural gas refineries, water pipelines, and nuclear power plants. Given that these systems are becoming increasingly connected, computer security is an essential requirement as compromises can result in consequences that translate into physical actions [3] and significant economic impacts [4] that threaten public health and safety [5]. Moreover, because the potential consequences are so great and these systems are remotely accessible due to increased interconnectivity, they become attractive targets for adversaries to exploit via computer networks. Several examples of attacks on such systems that have received a significant amount of attention include the Stuxnet attack [6], the U.S.-Canadian blackout of 2003 [7], the Ukraine blackout in 2015 [8], and attacks that target the control system data [9] itself. Improving the computer security of electrical power grids is the focus of our research.

The power grid is responsible for providing electricity to society, including homes, businesses, and a variety of mission critical systems such as hospitals, power plants, oil and gas refineries, water pipelines, financial systems and government institutions. The “smart grid” acts as an advanced power grid with upgrades that provide power distribution systems and consumers with improved reliability, efficiency, and resiliency [10]. Some of the upgrades include automated energy load balancing, real-time energy usage tracking and control, real-time monitoring of grid-wide power conditions, distributed energy resources, advanced end devices with two-way communications and improved processing capabilities. Advanced end devices, which are being integrated into smart grids, include Programmable Logic Controller (PLCs), Remote Telemetry Units (RTUs), Intelligent Electronic Devices (IEDs), and smart meters that are capable of controlling and performing physical actions such as opening and closing valves, monitoring remote real-time energy loads, monitoring local events such as voltage readings, and providing two-way communications for monitoring and billing, respectively. These new devices replace legacy devices that have been in place for decades that were not originally designed with security in mind since they were previously closed systems without external network connectivity. Although these new devices aid efficiency, they may create more avenues for attack from external sources.

Finally, control systems are often statically configured [11] over long periods of time and have predictable communication patterns [12]. After installation, control systems are often not replaced for decades. The static nature combined with remote accessibility of these systems creates an environment in which an adversary is well positioned to plan, craft, test and launch new attacks. Given that the power grid is actively being developed and advanced, the opportunity to incorporate novel security protections directly into the design phase of these systems is available and necessary. Of particular interest are defenses that can better avoid both damage and loss of availability, as previously documented in the power grid [5], to create a more resilient system during a remote attack over computer networks.

## 2. BACKGROUND

Artificial diversity is an active area of research with the goal of defending computer systems from remote attacks over computer networks. Artificial diversity within computer systems was initially inspired by the ability of the human body's natural immune system to defend against viruses through diversity [13]. Introducing artificial diversity into the Internet Protocol (IP) layer has been demonstrated to work within a software-defined network (SDN) environment [14]. Flows, based on incoming port, outgoing port, incoming media access control (MAC) and outgoing MAC, are introduced into software-defined switches from a controller system. The flows contain matching rules for each packet and are specified within the flow parameters. If a match is made within a packet, then the flow action is to rewrite source and destination IP addresses to random values. The packets are rewritten dynamically while they are in flight traversing each of the software-defined switches. Although applying artificial diversity on an SDN has been demonstrated, the effectiveness of such approaches has not been quantitatively measured. Furthermore, to our knowledge, the approach has not been deployed within an ICS setting, which differs substantially from traditional IT based systems.

It has also been demonstrated that IP randomization can be implemented through the Dynamic Host Configuration Protocol (DHCP) service that is responsible for automatically assigning IP addresses to hosts within the network [15]. Minor configuration modifications to the DHCP service can be made to specify the duration of each host's IP lease expiration time to effectively change IP addresses at user defined randomization intervals. However, this approach only considers long-lived Transmission Control Protocol (TCP) connections, otherwise disruptions in service will occur as the TCP connection will need to be re-established. Service interruptions within an ICS setting is not an option due to their high availability requirements. Quantifying the effectiveness of such approaches has also not been performed within an ICS setting outside of surveys [16] that evaluate Moving Target Defense (MTD) techniques within an IT setting, where IP randomization by itself was qualitatively ranked to have low-effectiveness with low operational costs. Also of note is that IP randomization approaches by themselves have been demonstrated to be defeated through traffic analysis where endpoints of the communication stream can be learned by a passive adversary observing and correlating traffic to individual endpoints [17].

Anonymization of network traffic is an active area of research with several implementations available in both the commercial and open source communities. MTD and anonymization are related in that they both have the goal of protecting attributes of a system from being discovered or understood. One of the early pioneering groups of anonymous communications describes the idea of onion routing [18] which is widely used today. This approach depends on the use of an overlay network made up of onion routers. The onion routers are responsible for cryptographically removing each layer of a packet, one at a time, to determine the next hop routing information to eventually forward each packet to their final destinations. The weaknesses of this solution are that side channel attacks exist and have been demonstrated to be susceptible to timing attacks [19], packet counting attacks [20], and intersection attacks [21] that can reveal the source and destination nodes of a communication stream.

The Onion Router (Tor) is one of the most popular and widely used implementations of onion routing with over 2.25 million users [22]. Tor is able to hide servers, hide services, operate over the TCP, anonymize web-browsing sessions, and is compatible with Socket Secure (SOCKS) based applications for secure communications between onion routers. However, it has been shown empirically with the aid of NetFlow data that Tor traffic can be de-anonymized with accuracy rates of 81.4% [23]. The results are achieved by correlating traffic between entry and exit points within the Tor network to determine the endpoints in communication with one another. Furthermore, Tor has an overhead associated with the requirement to encrypt traffic at each of the onion routers; this overhead would need to be limited within an ICS environment to meet the real-time constraints required of these systems. Similarly, garlic routing [24] combines and anonymizes multiple messages in a single packet but is also susceptible to the same attacks.

Overlay networks have similar goals as Tor with the goal of reducing the overhead associated with a Tor network. It has been shown that overlay networks can be used to mitigate Distributed Denial of Service (DDoS) attacks [25]. The overlay networks reroute traffic through a series of hops that change over time to prevent traffic analysis. In order for users to connect to the secure overlay network, they must first know and communicate with the secure overlay access points within the network. The required knowledge of the overlay systems prevents external adversaries from attacking end hosts on the network directly. This design can be improved by relaxing the requirement of hiding the secure overlay access points within the network from the adversary. If an adversary is able to obtain the locations of the overlay access points, then the security of this implementation breaks down and is no longer effective.

Steganography is typically used to hide and covertly communicate information between multiple parties within a network. The methods described in current literature [26] include the use of IP version 4 (IPv4) header fields and reordering IPsec packets to transmit information covertly. Although the focus of the steganography research is not on anonymizing endpoints, it can be used to pass control information to aid in anonymizing network traffic. The described approach would have to be refined to increase the amount of information (e.g, $log_2(n!)$ bits can be communicated through $n$ packets) that can be covertly communicated if significant information is desired to be exchanged. Steganography techniques have the potential to facilitate covert communication channels for MTD techniques to operate correctly but have not been applied in this fashion.

Transparently anonymizing IP based network traffic is a promising solution that leverages Virtual Private Networks (VPNs) and the Tor [27] service. The Tor service hides the user's true IP address by making use of a Virtual Anonymous Network (VAN) while the VPN provides the anonymous IP addresses. The challenge of this solution is the requirement that every host must possess client-side software and have a VPN cryptographic key installed. In practice, it would be infeasible for this approach to scale widely, especially within ICS environments where systems cannot afford any downtime to install and maintain the VPN client-side software and the cryptographic keys that would be necessary at each of the end devices. To reduce the burden on larger scale networks, it may be more effective to integrate this approach into the network level, as opposed to at every end device, using an SDN based approach. The goal of the research presented here is to provide a similar service within an ICS system with the ability to scale to a large number of devices without significant interruptions in communications.

## 3. THREAT MODEL

When evaluating the effectiveness of the ADDSec computer security protections, a threat model must first be established to define the adversaries' capabilities and the assumptions placed on the computer security protections. Additionally, the threat model describes the types of adversary capabilities that the security protections do not protect against. The focus of our research is on ICS based systems, including the network devices and the end devices in those environments. The threat model as a whole makes it clear when a particular protection is and is not an appropriate solution for a given environment or use case.

The network devices we examined were comprised of SDN capable switches and non-SDN capable switches. SDN capable switches provide administrators with an interface to dynamically program switching decisions and develop custom switching algorithms directly into the switch. The programmer can interact with a switch directly or through a controller system that also communicates with all of the SDN-based switches configured in the network. Traditional switches do not have the capability to be reprogrammed and are often times proprietary without a user Application Programming Interface (API) available. Traditional switches are also distributed in nature whereas SDN devices are managed through a centralized controller system that has a complete understanding of the network topology.

The MTD strategies we examined for our research were those that introduced randomization into IP addresses, application port numbers (TCP/UDP), inter-communicating network paths between systems, and the combination of the three techniques joined together. IP addresses, port numbers and communication path randomization are deployed as defensive strategies against a threat model in which adversaries possess the following capabilities:

- network access to the traditional network switches (non-SDN capable switches)
- access to the Switched Port Analyzer (SPAN) ports of all network switches outside of the edge switches
- ability to observe network traffic traversing traditional network switches (non-SDN capable switches)
- ability to collect network traffic traversing traditional network switches (non-SDN capable switches)
- ability to correlate end points within a network when in possession of packet captures
- ability to tap into network communication links outside of the edge switches connected to the end hosts
- ability to launch Denial of Service (DoS) attacks
- ability to launch Distributed DoS (DDoS) attacks
- access to the end devices behind the network switches

The threat model defined for our research considers adversaries with access to network traffic, either just captured or also removed, injected, or modified in real-time from the interior switches of the network. Using this data, we assumed that the adversary will attempt to correlate all endpoints given the packet traces captured. Another assumption is that there may be several adversaries collaborating and attacking the system together at the same time or separately. For the DoS and DDoS attacks mentioned above, we use metrics to provide the thresholds on how to detect those types of attacks given the number of probes observed within the network. We also allow the adversary access to the end devices which we depend on our machine learning algorithms to detect anomalous behavior.

Also of importance are the adversaries that this threat model, as applied towards IP, port and path randomization, does *not* protect against. This threat model does not consider adversaries that are in possession of any of the following capabilities:

- access to the edge switches that are directly connected to the end hosts
- administrative access to the SDN capable switches
- access to the SDN controller(s)

Adversaries with any of the capabilities listed above could gain privileges above the level at which the MTD techniques are effective. Adversaries with access to the edge switches would be able to observe and identify the end hosts whose information is being randomized. An adversary with administrative access to the SDN switches would be able to learn all flows installed on those switches which contain the information that is being randomized, in particular, the flows to randomize network paths and IP addresses on the edge switches. Adversaries with access to the controller would be able to modify and understand the random mappings since the controller(s) aggregate all of the network flows installed within the network.

The threat model and security protections developed as part of our research do not protect against every possible adversary, but do add an additional layer of defense. There are also a variety of other side channel attacks that may be available to extract information about the computer security defense such as attacking the random number generators that generate the random MTD mappings, but these additional side-channel attacks are outside the scope of this research and are areas for future research. We also assume that the software running in the SDN switches and controllers, such as the packet matching software, is secure. For the purposes of our research, the threat model is used as a basis so that the effectiveness of the MTD techniques can be quantitatively measured based on the criteria outlined in this section.

## 3.1. Operational Impact

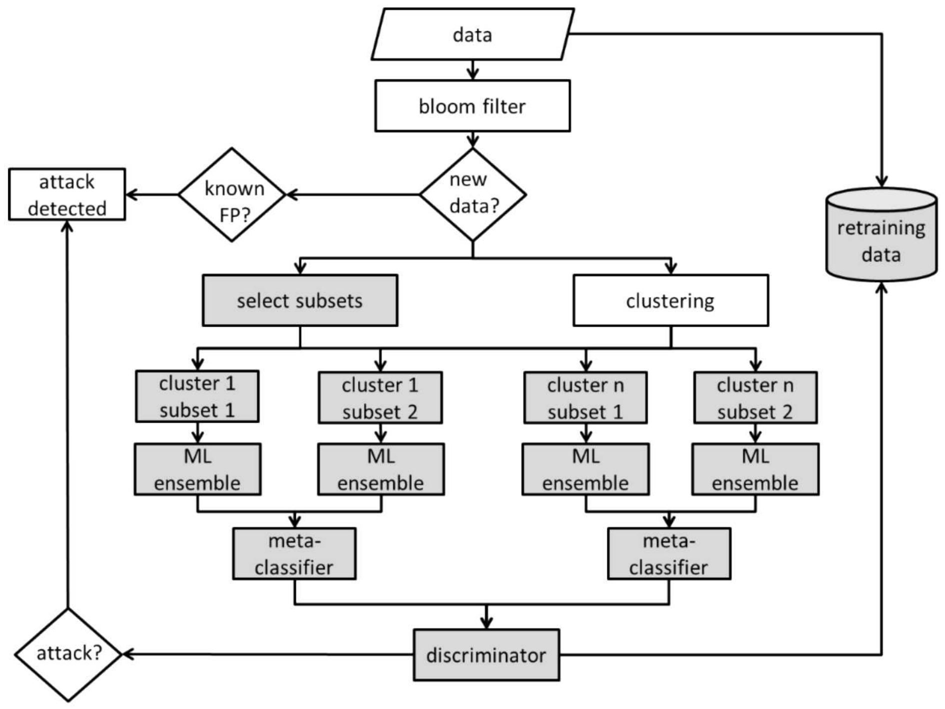
Each of the MTD techniques introduce additional layers of defense against an adversary who has the goal of gaining reconnaissance information from a network. Without these defenses, network access alone would be enough for an adversary to learn the IP addresses, port numbers and communication patterns by passively observing the network packets traversing the network. After observing this information passively, the adversary can then plan, research and launch attacks against the IP addresses or services at a later time. One of the goals of our research is to introduce delays into the reconnaissance phase of an attack and measure those delays to determine the effectiveness of each technique. Ideally, the delays will be a significant enough amount of time to detect or deter an adversary from launching a successful attack. The real-time constraints and requirements of ICS environments are also taken into account and those impacts on the operational network are measured. The tradeoff between security and usability as well as finding a balance between maximizing the adversarial workload while minimizing the impact to both the defender and to the operational network is the focus of our research. Some of the operational metrics measured and captured include latency, throughput, bandwidth, network utilization, CPU utilization, and memory utilization.

## 4. DYNAMIC DEFENSE

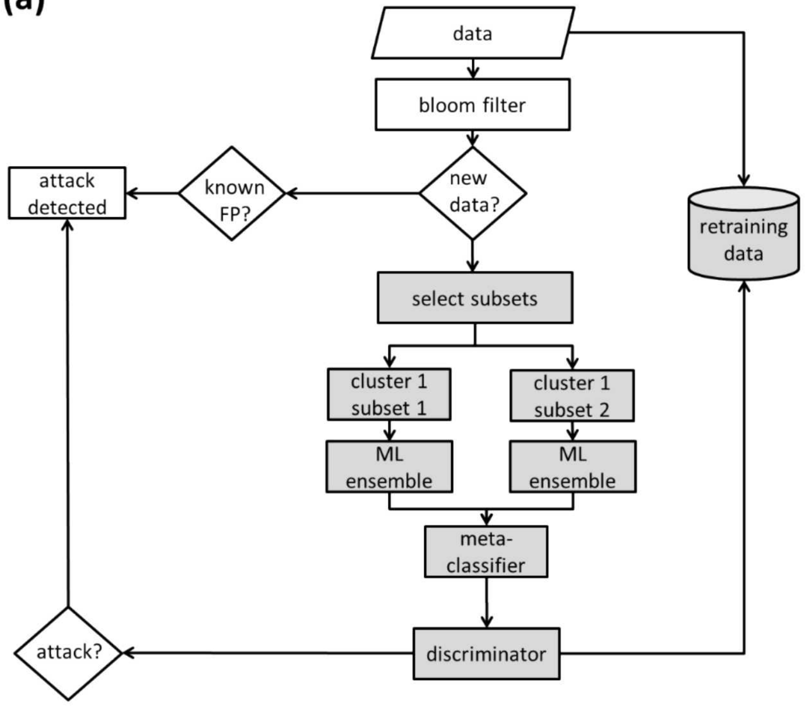
Dynamically defending a network against an active attack has many similarities to a chess master. Chess masters are amongst the best chess players in the world and often succeed by quickly recognizing patterns combined with strategically mastering the opening and middle moves of a game. Once a pattern is recognized, the chess master draws from the experience gained from many games to deploy appropriate response strategies. The same strategies apply to successfully defending critical infrastructure systems. Critical Infrastructure control systems often harness unpatched (legacy and modern) systems that allow easy access to our Nation's most critical assets making them attractive and easy targets for cyber attack. On the other hand, critical infrastructure systems communicate and execute programs in fairly predictable patterns relative to traditional IT based systems lending themselves well to dynamic defense strategies. Specifically, the dynamic defense project aims to better secure control systems by recognizing and dynamically defending them against active attacks.

### 4.1. Framework

Currently, it is extremely difficult to detect an attack until it is too late. We have developed a machine learning framework that recognizes active attack patterns in near real time. This framework, shown in Figure 1, is capable of recognizing known attack vectors and also of generalizing knowledge from known attacks to recognize new attacks. In our implementation, we respond to these attacks by transforming our networks and applications into moving targets by applying the techniques implemented from the network randomization project. The framework itself is dynamic, allowing the specifics of the detection mechanism to change over time. Each component in our framework is described in the following sections.



**(a)**



**(b)**

**Figure 1.** Our framework developed to classify traffic into anomalous or normal categories.

### 4.1.1. *Raw Data*

Initially, features are extracted from raw input data. For our initial experiments, we employed the NSL-KDD dataset, which consists of features extracted from network traffic [47]. This dataset is similar to the famous KDD Cup '99 dataset, but it corrects a few problems from the original KDD Cup '99 data, and is also a multi-class dataset. While this dataset is nearly 20 years old, it is the only publically available multi-class cybersecurity dataset that we are aware of. Consequently, we use it to develop or multiclass dynamic defense solution.

The vectors in this dataset consist of the same 41 features used in the original KDD Cup '99 dataset. The data consists of 39 different types of attacks in four different classes (Denial of Service, User to Root, Remote to Local, Probing), plus normal data.

Amongst these features are:

- *duration*: length (number of seconds) of the connection
- *src_bytes*: number of data bytes from source to destination
- *dst_bytes*: number of data bytes from destination to source
- *count*: number of connections to the same host as the current connection in the past two seconds
- *dst_host_count*: among the past 100 connections whose destination IP address is the same to that of the current connection, the number of connections whose source IP address is also the same to that of the current connection
- *dst_host_srv_count*: the number of connections in the dst_host_count feature whose service type is also the same to that of the current connection
- *same_srv_rate*: % of connections in the count feature to the same service
- *srv_serror_rate*: % of connections whose service type is the same to that of the current connection in the past two seconds that have "SYN" errors

and features derived from these:

- total_bytes: src_bytes + dst_bytes
- log10(*src_bytes*)
- log10(*dst_bytes*)
- *src_bytes / duration*
- *dst_bytes / duration*
- *total_bytes / duration*
- *log_duration*: log10(*duration*)
- *count / duration*
- *dst_host_count / duration*
- *dst_host_srv_count / duration*
- *dst_bytes * dst_host_count:*
- *dst_bytes * dst_host_srv_count*
- *same_srv_rate * srv_serror_rate*

- *same_srv_rate * dst_host_count*
- *same_srv_rate * dst_host_srv_count*
- *srv_serror_rate * dst_host_srv_count*
- *dst_host_count * dst_host_srv_count*

Inclusion of these features improves the performance of the machine learning algorithms. A description of the features is available from [29]. In addition to these features, we also generate some cross-product terms by multiplying one feature with another. This can result in a large number of features, so we use feature selection in our preprocessing step to reduce the number of features provided to the machine learning algorithms.

Of course, the framework can be used with other types of data as well. In particular, host based features, such as system performance statistics and statistics derived from sequences of system calls, can also be used [30]. We represent the service and flag features with a one-hot binary encoding.

This raw data set is used to train our machine learning algorithms for future classification of unknown traffic and should be representative of what is observed in normal operations. Although we used a readily available dataset for our proof-of-concept implementation, actual traffic of the system where the dynamic defense technology will be deployed should be used for training the machine learning algorithms.

### *4.1.2. Bloom Filters*

After feature extraction we process the data with a sequence of Bloom filters. Bloom filters are a probabilistic method for determining the newness of data. A Bloom filter consist of an array in which all values are initialized to zero. When data is received by the Bloom filter, the data is first hashed by several hash functions. The hashed values are then used as indices into the Bloom filter array. At each index pointed to by a hashed value, the value in the Bloom filter is set to 1. With 100% probability old data will have a '1' at each index, while with high probability that is adjustable by controlling the parameters of the Bloom filter, the new data will have a '0' at one or more indices [31].

We use Bloom filters for data reduction and to help control the false positive and false negative rates. The first Bloom filter separates new data from old so that we only have to analyze new data with the machine learning models. Old data is then sent to a Bloom filter containing feature vectors from false positives. If the data is found in this filter, then it is labeled as being normal data. If the data is not in the false positives Bloom filter, then it is processed by a Bloom filter containing feature vectors from attack data. If the data is found in this filter then it is labeled as being attack data, otherwise it is labeled as being normal data. The false positive and attack Bloom filters are initially populated with known vectors, with the class label removed, from a labeled training set. During operation, all feature vectors are added to the initial Bloom filter, and any vectors classified as attacks are added to the attack Bloom filter, again with the class labels removed. In practice, we envision that, in addition to these automated procedures, an analyst could maintain the false positives and attacks Bloom filters. In support of this, our framework adds all feature vectors classified as attacks to a database. In practice, the raw data associated with these vectors would also need to be stored.

### 4.1.3. *Softmax Scaling*

Next, all new data is softmax scaled as a preprocessing step. Softmax scaling is a normalization approach that does minimal harm to the information content of the input dataset, Softmax scaling compresses all numerical values to the range [0,1].The transformation is linear at the mean but has smooth nonlinearities at both extremes of the range. This helps the machine learning models process data that is outside of the ranges observed during training [31].The normalized data, $v_n$, is calculated as:

$$v_n = \frac{1}{1 + e^{-v_t}} \tag{1}$$

where

$$v_t = \frac{v_{i\neg\dagger} - mean(v_i)}{std\neg\dagger\, dev(v_i) \big/ 2\text{œÄ}} \tag{2}$$

and $v_i$ is the unscaled data. There is one $v_t$ for each continuous feature, and new $v_t$ values are calculated during retraining cycle. The softmax scaled data is then provided as input for the Bloom filters.

### 4.1.4. *Generating Retraining Datasets*

Our framework generates datasets for periodic retraining. Retraining data consists of a mix of old data from previous training or retraining datasets, and of newer data processed since the last retraining cycle. In our current implementation retraining occurs after 200,000 new feature vectors have been processed. Of these new feature vectors, approximately 6% are considered for inclusion in the retraining dataset. Of these 6%, with 80% probability we add the vector to the retraining dataset. The remaining 20% of the time we uniformly at random choose a feature vector from the previous training or retraining dataset for inclusion. Additionally, we target our retraining datasets to consist of an even balance of 5000 attack vectors and 5000 normal vectors. To achieve this balance, we check for deviations from it immediately before retraining. If more than 60% of the retraining dataset has the same class label then we remove some vectors and replace them with vectors from the previous training or retraining dataset to achieve a balance. Since labeled datasets are required for training, we label the new data with the classification provided by our models.

### 4.1.5. *Clustering*

Optionally, we begin by assigning each feature vector to one of *m* clusters, where *m* is user configurable. We do this with a k-nearest neighbors classifier with *k=1*, so that each feature vector is assigned to the class of its single nearest neighbor from the training data. We then train an ensemble of multi-class classifiers for each cluster, as described in the following sections. We term this architecture as our “cluster first” architecture. This is shown in Figure 1a. We can also remove the initial clustering, but leave the rest of the system the same. This is shown in Figure 1b.

This multi-class classification allows us to automatically choose different responses to different types of detected anomalous behavior. For example, some network randomization approaches may be more effective at disrupting some types of attacks than others.

### 4.1.6. Feature Subsetting

The feature vectors are split into two subsets by uniformly at random assigning each feature to one or both subsets. The one-hot encoded representations of the service and flag features are included in both subsets. The benefit to splitting the feature vectors into two feature sets is to attempt to provide independence between the models trained on the different feature sets. When the models trained on the different feature sets are later combined into an ensemble, if the classification errors of the subsets of models are uncorrelated, due to their independent feature sets, then the final ensemble can have better performance than any of the individual classifiers [33].

### 4.1.7. Boosted Classifiers

We train an ensemble of classifiers on each of the feature subsets. Each ensemble contains one each of the Naïve Bayes, logistic regression, support vector machine (SVM), and random forest classifiers [34,35]. We name these “level 1 classifiers”. We use the Orange Data Mining Toolbox for our implementations of the models [36]. Each of these classifiers has a number of configurable hyper-parameters. For instance, in Naïve Bayes we modify the probability estimator used for estimating prior class probabilities amongst m-estimates, Laplace estimation, and relative frequency estimation, and can also choose whether to tune the classification threshold for choosing between classes. In SVM we can select amongst different SVM types, kernel types, various parameters for the different SVM types and kernels, and termination criteria. In logistic regression we modify the solver type and regularization parameter, and with random forests we vary the number of trees in the forest and the number of features to consider when determining where to split the nodes during tree induction [36].

We perform a random search to optimize selection of these hyper-parameters [37]. In each iteration of the search we randomly assign values to the hyper-parameters for the classifier under consideration, and then create a boosted learner consisting of 30 learners with these hyper-parameters [40]. The resulting boosted learner is then trained on half of the training data and tested on the other half. The division of the training data is random and differs for each of the machine learning techniques. The learners are evaluated using Matthew’s correlation coefficient (MCC) [38]. In terms of true positives (TP), true negatives (TN), false positives (FN), and false negatives (FN), the MCC is defined as:

$$MCC = \frac{TPv\acute{o}TN - FPv\acute{o}FN}{(TP + FP)(TP + FN)(TN + FP)(TN + FN)} \tag{3}$$

and has range [-1,1] with a score of 1 indicating perfect performance, 0 indicating no better than chance, and -1 indicating complete inaccuracy in prediction. The random search is terminated if an iteration's score differs from the previous maximum by less than 0.001 and the maximum score exceeds 0.7, or when a maximum of 15 iterations are reached. Experimentally, we find there to be little gain from introducing additional iterations to the search.

We always seek to optimize the performance of the ensemble, rather than that of the individual classifiers [39]. As such, the optimization begins with an empty ensemble and searches for an optimal hyper-parameter selection for the Naïve Bayes classifier. Then, the logistic regression, SVM, and random forest classifiers are added, in turn, to the ensemble. Combining heterogeneous classifiers into an ensemble in this way has been shown to increase prediction accuracy [41].

#### *4.1.8. Boosted Meta-Classifiers*

The classification probabilities from our two level-1 classifiers are input to an ensemble of meta-classifiers. During training we randomly choose to include between 8 and 15 classifiers in our ensemble of meta-classifiers. Each of the meta-classifiers is randomly chosen to be a Naïve Bayes, logistic regression, SVM, or random forest. Each of the meta-classifiers takes as input the classification probabilities from a random selection of two to six of the level-1 classifiers. As with the level-1 classifiers, we perform a random search for hyper-parameter optimization, form a boosted learner containing 30 learners with these hyper-parameters, and use MCC to score the models.

#### *4.1.9. Boosted Discriminator*

The classification probabilities from the two blended level-1 ensembles and from each of the meta-classifiers are input to a random forest for final classification. As always, we perform a random search for hyper-parameter optimization and form a boosted collection of 30 learners with these hyper-parameters. During testing, the output from this discriminator is used as the classification for each feature vector. These classifications are then used to label the (unscaled) testing data so that it can be incorporated into the retraining dataset and, as necessary, added to the attacks Bloom filter.

## 3.1 Dynamic Aspects

There are many dynamic aspects of our framework. For example, one can randomize the selection of subsets of features for the level-1 ensembles. The number of level-1 ensembles can also be varied. We can adjust the number and type of classifiers in the level-1 ensembles and also the number and types of meta-classifiers. The selection of retraining data is randomized, and the balance of new and older data in the retraining datasets can be varied. We can also vary the frequency of retraining. We can also train several distinct copies of the framework and randomly choose which of the copies to use to classify incoming data. All of this randomization should frustrate adversaries that attempt to inject malicious data to train the models to classify adversarial data as being normal. The dynamic aspects are shaded in gray in Figure 1.

## 4.2. Results

Results from testing our framework on the NSL-KDD data and comparison to results from the literature are presented in Table 1 [55-62]. We present results for both our cluster-first and our regular architectures. Disparity in the number of samples from different class types remains a problem with the NSL-KDD dataset. The relative sparsity of Probe, R2L, and U2R classes biases classifiers toward detecting the Normal and DoS classes. This is evidenced by high accuracy for the Probe, R2L, and U2R classes even when all vectors are classified as DoS or Normal. Finally, we also note that our interest was in developing a multiclass classifier; no particular effort was made to optimize our classifier for this dataset, and so it is not surprising that our results are not as impressive as some of those in the literature.

**Table 1:** Results from testing our 5-class model on the NSL-KDD dataset, and comparison to results from the literature.

| **Classifier** | **Class Type** | **Recall** | **Accuracy** | **FPR** |
|---|---|---|---|---|
| **Ours (cluster first)** | Normal | 0.897 | .864 | 0.160 |
| | DoS | 0.936 | 0.819 | 0.238 |
| | Probe | 0.098 | 0.861 | 0.047 |
| | R2L | 0.0 | 0.872 | 0.0002 |
| | U2R | 0.0 | 0.997 | 0.0002 |
| **Ours** | Normal | 0.958 | 0.685 | 0.524 |
| | DoS | 0.792 | 0.905 | 0.039 |
| | Probe | 0 | 0.897 | 0 |
| | R2L | 0 | 0.871 | 0 |
| | U2R | 0 | 0.997 | 0 |
| Dhanabal J48 [55] | Normal | | 0.998 | |
| | DoS | | 0.991 | |
| | Probe | | 0.989 | |
| | R2L | | 0.987 | |
| | U2R | | 0.979 | |
| Panda [57] | | | 0.965 | 0.30 |
| Tavallaee J48 [58] | | | 0.938 | |
| Revathi Random Forest[59] | Normal | | 0.998 | |
| | DoS | | 0.991 | |
| | Probe | | 0.989 | |
| | R2L | | 0.987 | |
| | U2R | | 0.979 | |
| Ingre neural network[60] | Normal | | | |
| | DoS | 0.777 | 0.926 | 0.013 |
| | Probe | 0.766 | 0.950 | 0.0236 |
| | R2L | 0.346 | 0.908 | 0.0019 |
| | U2R | 0.105 | 0.990 | 0.0006 |
| Mukherjee [61] | Normal | 0.97 | | |

|  |  |  |  |  |
|---|---|---|---|---|
|  | DoS | 0.987 |  |  |
|  | Probe | 0.988 |  |  |
|  | R2L | 0.961 |  |  |
|  | U2R | 0.64 |  |  |
| Heba [62] | Normal |  | 0.995 |  |
|  | DoS |  | 0.975 |  |
|  | Probe |  | 0.928 |  |
|  | R2L |  | 0.813 |  |
|  | U2R |  | 0.866 |  |

## 4.3. Streaming Implementation

We modified the previously described Dynamic Defense system to create a streaming implementation for use in testbed and operational settings. Rather than ingesting existing feature vectors, the streaming implementation creates its own feature vectors by sampling network and host data, extracting relevant statistics and features from those data, and then processing the resulting feature vectors.

Our streaming implementation was built in python for Linux operating systems and has been tested on Ubuntu 12.04 and 16.04. Implementations for other operating systems are possible, but may require modifications to the data collection tools. Our implementation uses tcpdump and bro to extract the network features described in section 4.1.1[48,49]. These features are captured over one second intervals. We also capture host features for our streaming implementation. For this, we use strace to record system calls, sar to capture system loads, and *etime* to capture the execution time of processes [50,51,52].

For the system call features, we capture logs of all system calls during a one second window. The training data is used to find sequences of system call operations and sequences of the results of those operations on a process by process basis. These sequences are stored in trees. Then, during testing we find the Hamming distances between the test sequences and those from the training set. We experimentally found that the 12 most frequently called processes account for most of the system operations. For features, we use the minimum Hamming distance, number of new processes observed in the trace, percentage of system calls from processes that are not in the top 12 processes, percentage of system calls from new processes, and the percentage of sequences that were not present in the training set. We are motivated to use system calls by past research in the area [53]. We use the sar utility to capture system loads. Similar types of data have been used by other researchers to detect computer misuse and abnormal usage [54, 55]. For this, we run the sar command:

```
    sar -A 1 1
```

which captures all of the system activity statistics over a single one second interval. We then parse the data to extract the system usage data, and append this to the system call features from strace and the network features. The result is a feature vector representing one second of host and network activity for the endpoint.

We use the command:

```
    ps -U root -o etime= > elapsed_time.txt
```

to find the execution time of running processes. We then find the minimum, maximum, mean, median, and standard deviations of these times and use them as features.

In total, we capture 292 features. Each feature vector represents one second of system behavior, although the length of this window can be set to a user configurable t seconds. We capture data over n such windows and collect all of the resulting feature vectors in a file. Then, we classify each of the n features. This amortizes the memory overhead across several features, but introduces a delay of up to (n-1)t seconds for detecting anomalous behavior.

### *4.3.1. Feature Selection*

To improve classifier performance and reduce computational burden, we perform a feature selection step to identify those features that are most useful. For this, we begin by applying several feature scoring methods (information gain, information gain ratio, Gini coefficient[*], chi-squared, and reliefF[†]) to the original 292 features. We then find the top 20 features according to each of these five measures, and keep all of the resulting unique features. Then, we begin a feature generation step. Here, we generate new features by multiplying each of the surviving features by each of the other features. We add these to our feature vector, and then we repeat the feature selection step, again finding the top 20 features according to each of the five measures and retaining the resulting unique features. We then generate additional features from these by squaring each of the features and appending the results to our feature vectors. Finally, we repeat the feature scoring and down-selection step a third time. This results in approximately 50 features, consisting of a mix of the captured features and the derived (cross-product and squared) features to use for classification. These feature selection and generation steps are initially performed off-line, and then we update our feature collection code to generate the necessary derived features and form the reduced feature vectors.

### *4.3.2. Multiclass Classification*

We support multiclass classification to differentiate between different types of attacks. This allows us to automatically respond to different attacks in different ways. Our multiclass classifiers use the same algorithms (naïve Bayes, random forest, support vector machine, logistic regression) as in binary classification, but instead of determining only whether a feature vector represents normal or attack behavior, they now differentiate between different classes of attacks.

[*] Gini, C. (1909). "Concentration and dependency ratios" (in Italian). English translation in *Rivista di Politica Economica*, **87**(1997), 769–789.

[†] Kononenko, Igor et al. Overcoming the myopia of inductive learning algorithms with RELIEFF (1997), Applied Intelligence, 7(1), p39-55

## 5. MOVING TARGET DEFENSE

We implemented and evaluated three MTD techniques to quantify their effectiveness, both individually and in combination with one another. IP modification has the goal of creating uncertainty and increasing the difficulty for an adversary to discover and track the systems existing within a computer network. IP randomization is implemented using an SDN based approach with the flow rules installed at each switch controlling the randomization intervals. Port modification has the goal of hiding the services offered by a system. Port randomization is implemented as a standalone solution that is placed at each individual end point within the network. Path modification supports both of the previous two goals and also has a third goal of preventing an adversary from learning the end points of a network by observing traffic and performing analysis to correlate the end systems that are actively communicating with one another. Path randomization has been implemented in an SDN based approach with the flow rules controlling the next hop switches in the network. Each technique provides its own sets of benefits and are described in more detail in the sections that follow.

### 5.1. IP Randomization

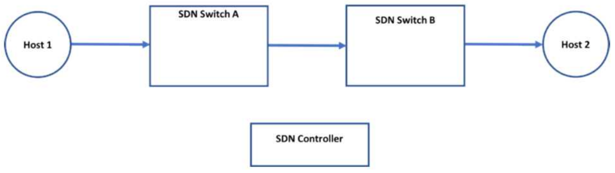
The first MTD defense we developed focuses on randomizing IP addresses at user configurable frequencies to evade adversarial discovery. The randomization algorithms reside at the network layer, transparent to the end devices themselves, for both usability and scalability purposes. Improved usability comes from the fact that the randomization algorithms are managed at the network level and do not need to be deployed at every end device. The algorithms are built into the SDN fabric, which consists of several SDN-capable switches and a management controller system. The management burden is reduced since the network security personnel would only have to maintain the network layer devices as opposed to all of the end devices of the network (which typically outnumber the network layer devices by far). The SDN architecture provides a scalable solution where any new end device introduced into the network will automatically have the IP randomization MTD defense activated and enabled, without the end user necessarily having any knowledge that the MTD defense deployed. Additionally, there are typically far fewer end devices than there are network based devices within a network.

An SDN based approach is also used so that routing and switching logic can be customized to control the frequency at which source and destination IP addresses are randomized. The customized logic can also account for the periods of time when a packet is traversing the network and new randomized source and destination IP addresses are installed on each of the switches in the network. The random IP address mappings are programmed by flow rules that are managed by a centralized controller. Each flow rule has a “match” specification and an associated “action” to perform depending on if the match criteria is satisfied. The flow rules for this implementation match a packet based on a combination of the source IP address, the destination IP address, and the incoming physical port of the switch that the packet was received on. If the incoming physical port and source IP address that the packet was received on corresponds to a host that is directly connected to the switch, then the action taken within the flow rule is to rewrite the source and destination IP addresses with a set of newly generated random source and destination IP addresses. Otherwise if the host is not directly connected to the switch (but rather to an interior switch interface), the packet is forwarded to the next hop switch. The location of the next hop switch is specified within the flow rules that matched the randomized source and destination IP addresses for that particular MTD reconfiguration interval. The random mappings are communicated to the SDN controller via a Python wrapper script that updates the mappings based on predefined user configurable randomization intervals desired.

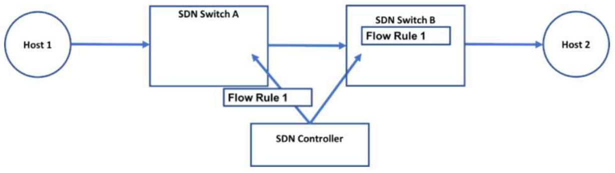
Once the packet reaches the edge switch that is directly connected to the destination host, the original source and destination IP addresses are restored within the packet back to the original source and destination IP addresses. The result of this approach is that an adversary passively observing traffic, on an interior non-SDN capable network switch, will no longer automatically learn the true IP addresses of the end hosts simply by observing network traffic passively; in this scenario, the adversary would instead observe a pair of pseudorandom IP addresses traversing the network. The pseudorandom IP addresses that are managed by the controller are continuously changing at user configurable time intervals and the random mappings are generated using a pseudorandom number generator. For this implementation, the entire 32-bit IP address is randomized since we are analyzing a flat network that contains only layer 2 switches. If routers are included in the topology, a quick solution to deploy this MTD technique would be to only modify the host bits of the IP address. This solution would allow layer 3 devices to appropriately route packets without having to modify the underlying routing protocols within those devices. A similar approach could have also randomized MAC addresses or even set all MAC addresses to the same value since the match criteria of the SDN flow rule does not depend on MAC addresses. Manipulation of MAC addresses was not performed in this research but can easily be adapted to do so. We have developed a prototype that has been tested on topologies of over 300 end devices operating within a virtualized environment using mininet [63]. Original development of the SDN algorithms were completed through the POX [44] controller, and now include the Ryu [71] and OpenDaylight [64] controllers.

The algorithms to randomize the IP addresses had to be slightly modified in practice to work correctly and avoid packet losses. The OpenDaylight controller [64] is an open source model-driven SDN controller which communicates to SDN capable devices through the OpenFlow protocol. The OpenFlow protocol [65] is a standardized interface with an internal table that manages communications between controllers and the flow rules of how to switch packets. OpenFlow version 1.3 supports axillary connections to the controller(s), functions to control the rates of packets through meters per flow, and cookies can be added as identifiers to each flow rule to help add, update, and delete flow rules as desired. Most vendors developing SDN capable switches support the OpenFlow 1.3 protocol.

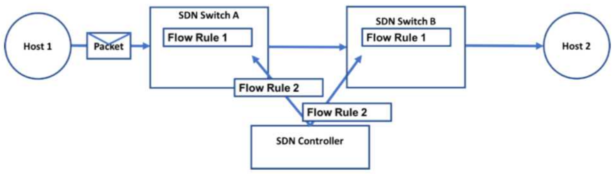
During our experiment, it was observed that when using the OpenDaylight implementation packets would occasionally be dropped. The dropped packets occurred in scenarios where new flow rules were installed slightly before a packet arrived at the last hop SDN switch, which is the switch that is responsible for delivering the packet to the final destination system. This scenario is shown in Figure 2 (a) - (g). The *hardtimeout* parameter within the OpenFlow protocol is used to configure the time period in which a flow rule is active. The cause for the dropped packets is that the SDN switches are not perfectly synchronized in time with each other when new packets flow rules are installed at each of the switches. These minor variations in flow rule installation time cause the flow rules to expire at slightly different times through the *hardtimeout* value specified. The small window of time where two switches may not have a consistent view of the network, in terms of the active flow rules installed, needed to be addressed and corrected for the solution to be usable in practice.



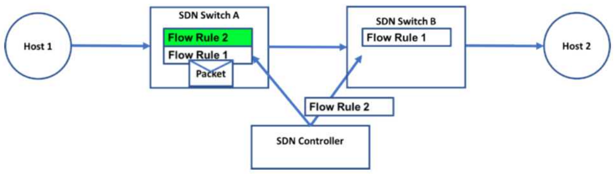
(b) Two hosts would like to communicate over an SDN network



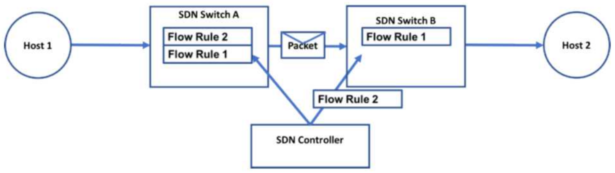
(a) The controller installs Flow Rule 1 on SDN switches A and B to randomize IP addresses between Host 1 and Host 2



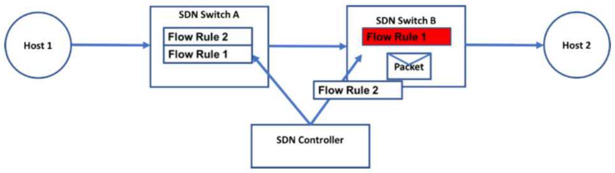
(d) After a period of time, new randomized source and destination IP addresses are generated to send to each of the SDN switches A and B labeled as Flow Rule 2. At the same time Host 1 sends a packet to Host 2



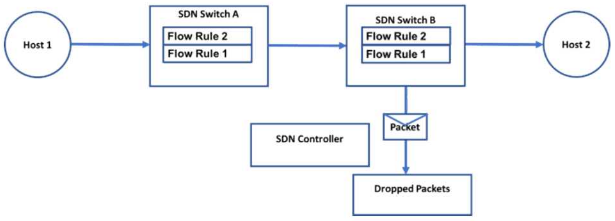
(c) The new flow rule is installed on the SDN Switch A but has not reached the SDN Switch B due to network congestion. The packet from Host 1 is matched with Flow Rule 2 installed at SDN Switch A



(f) The source and destination IP addresses are rewritten according to Flow Rule 2 and are forwarded to SDN switch 2. Flow Rule 2 has still not yet reached SDN Switch B



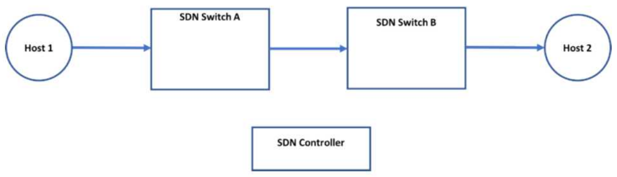
(e) The packet is received on SDN Switch B and since Flow Rule 2 has still not yet reached SDN Switch B, there is no match with Flow Rule 1 which is the only rule installed



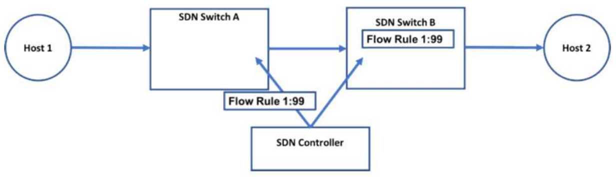
(g) The packet is dropped since Flow Rule 2 has not yet reached SDN Switch B. Flow Rule 2 is now installed on SDN Switch B, but it is too late

**Figures 2 (a)-(g).** The sequence of events that cause packets to be dropped when randomizing source and destination IP addresses within an SDN setting.

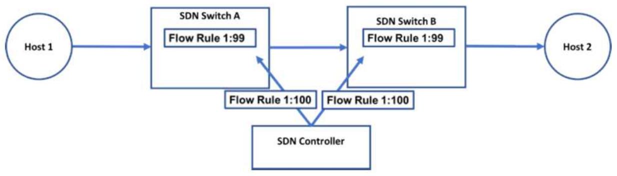
To correct this issue, new flow rules are initialized and installed on each of the SDN-capable switches by the controller. The old flow rules are configured to overlap with the new flow rules at a user configurable period of time, 3 seconds in this case. When new flow rules are installed at each of the SDN-capable switches, they are installed with a lower *priority* field, within the OpenFlow protocol, than the existing flow rule previously installed. After the controller sends the new flow rules to be installed at each of the switches, the controller receives an acknowledgment from each switch that the new flow rule was accepted and installed. The new flow rules, however, are not actively matched at this point since they have a lower *priority* than the previous flow rules installed. The next step is to swap the flow *priorities* of the old and new flow rules that are installed at each of the switches. Finally, the old flow rules will eventually expire before the new flow rules at specific time periods that depend on the user configurable setting of the overlap window. The process of SDN flow rule installation to prevent packets from being dropped is shown in Figure 3 (a) - (l). This solution allows old flow rules to deliver packets to the destination device without worrying about the slight variations in timing when the flow rules are installed and expired. The *priority* field is used to determine when flow rules become active in matching packets to randomized source and destination IP addresses.



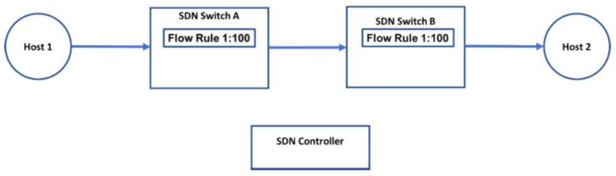
(a) Two hosts would like to communicate over an SDN network



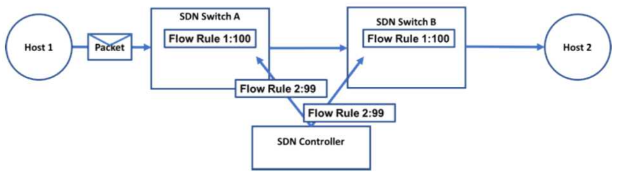
(b) The controller installs Flow Rule 1 with priority 99 on SDN switches A and B



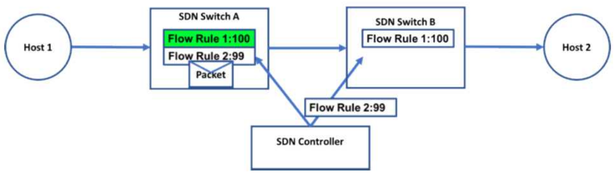
(c) Immediately after Flow Rule 1 is installed, Flow Rule 1 is updated to have priority 100



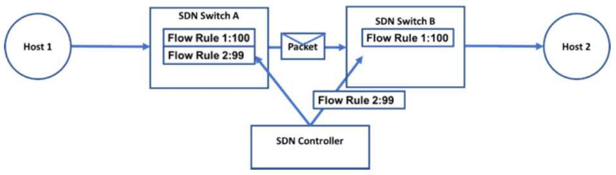
(d) Flow Rule 1 is updated on both SDN Switch A and SDN Switch B to have a priority of 100



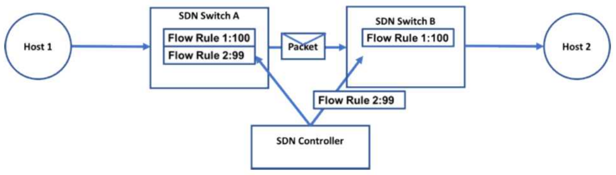
(e) After a given period of time, the controller pushes out Flow Rule 2 with priority 99 and new randomized source and destination IP addresses. Host 1 sends a packet to Host 2



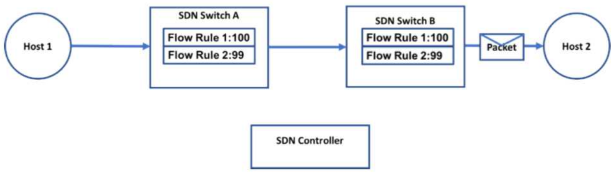
(f) The packet is received on SDN Switch A and matched on Flow Rule 1 since it has a higher priority. Flow Rule 2 has reached SDN Switch A but not SDN Switch B



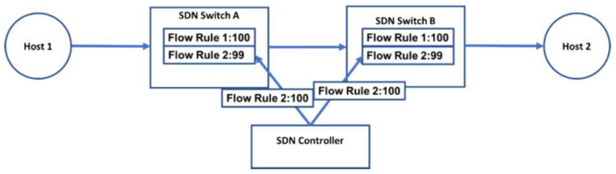
(g) Flow Rule 2 with priority 99 is installed on only SDN Switch A so far. The packet is forwarded to SDN Switch B



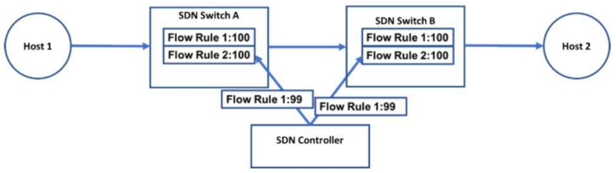
(h) SDN Switch B receives and matches the packet against Flow Rule 1 since Flow Rule 2 has not been received



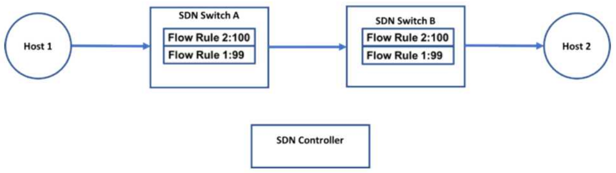
(i) Flow Rule 2 is installed on SDN Switch A and B with priority 99



(j) The SDN Controller immediately updates the priority of Flow Rule 2 to 100



(l) The SDN Controller updates the priority of Flow Rule 1 to 99. Flow Rule 2 takes priority



(k) Flow Rule 1 is updated to priority 99 and Flow Rule 2 has priority 100

**Figures 3 (a)-(l).** The sequence of events to correct dropped packets from occurring when randomizing source and destination IP addresses within an SDN setting.

## 5.2. Port Randomization

The second MTD defense developed randomizes the application port numbers of services running on each of the end devices within the network. Port randomization was implemented using the *iptables* packet filtering tool, a tool that is typically used to configure and enforce firewall policies on a system. The *iptables* tool has several rule chains, each of which are triggered at different points within the network stack when evaluating packets that are sent and received. The rule chain used to randomize port numbers is the same rule chain that manages firewall rules for network address translation (NAT) [48] based networks. The NAT rule chain evaluates packets just before and just after they exit and enter the network interface card, respectively. If a match is made within the *iptables* rule, then the specified actions are performed on the packet. The actions for port randomization involve rewriting the source and destination port numbers, based on a pseudorandom number generator that is seeded based on time, to random values.

Each host in the network is initiated with the same agent software installed which are all synchronized in time with an Network Time Protocol (NTP) server so that they can generate the same pseudorandom port mappings. Alternative methods exist to synchronize end hosts within the network, but the approach to synchronize on time was chosen to conceptually evaluate host based MTD solutions. The frequency of creating new random mappings is user configurable and, in this case, has been chosen to regenerate new mappings every one second. An effect of applying this MTD strategy is that an adversary can no longer automatically learn the services running on a network by passively observing network traffic alone. To defeat this defense, adversaries would be required to have the ability to perform deep packet inspection to identify the protocol and service offered by the end devices within the configured randomization interval. This does not completely eliminate all threats, but does add an additional layer of defense to delay an adversary or alert an operator if previous port mappings that have expired are actively being scanned by an adversary.

## 5.3. Path Randomization

The third technique developed randomizes the communication paths that packets take within a network. This approach utilizes SDN-capable switches to install flow rules that specify pseudorandom paths into each of the SDN-capable switches. The flow rules are installed into each of the network devices and are replaced with new random mappings at user configurable time periods. The pseudorandom paths are generated by the controller which has complete knowledge of the network topology. The controller first generates all possible paths between all possible pairs of systems when the network is being initialized. After initialization is complete, a randomly-selected path is chosen by the controller and the flow rules associated to that path are installed at each of the SDN switches to appropriately route traffic at user configurable periods.

The effectiveness of this technique is based on the amount of entropy available within the network (i.e. the number of possible paths within the network in this case). To achieve a high level of entropy, the number of SDN-capable switches should be increased so that there are more options of paths that are available within the network. As the number of SDN capable switches grows, so does the number of possible paths within the randomized pool, making it more difficult to correlate end points within a network given the additional entropy introduced. The number of hops should be considered as the source of entropy for this strategy, so the controller or switches themselves are not compromised. The greater the entropy and the number of hops between endpoints, the greater the workload that is placed on an adversary who has the goal of correlating and identifying ingress and egress communication patterns in the network.

Without the addition of path randomization, traffic analysis is a concern for each of the approaches described thus far. A passive adversary observing traffic may still be able to determine the end points in communication by analyzing each of the network communication streams. We have developed two proof-of-concepts, one using a POX controller and another using an OpenDaylight controller, that periodically randomize communication paths using an SDN network. POX [44] is a controller written in the Python programming language which supports the OpenFlow version 1.0 protocol. OpenFlow version 1.0 is very useful for modifying packets through flow rules which is needed to randomize IP addresses and paths within the network. However, later in our work when we discuss fault tolerant algorithms, the requirement for the SDN switches to communicate with multiple controllers is necessary, which OpenFlow version 1.0 does not support.

The algorithm to randomized communication paths assumes that the SDN controller has a priori knowledge of the network topology and that the systems wishing to communicate are part of a connected graph [48]. If the assumptions are valid, then all possible paths between each pair of systems within the network are enumerated and stored within a hash table stored at the controllers for quick path lookups. To determine the communication paths, the controller selects a random path between two endpoints and installs the appropriate flow rules at each of the intermediate switches to enforce that packets will traverse the random communication path currently selected. The high level algorithm is shown below.

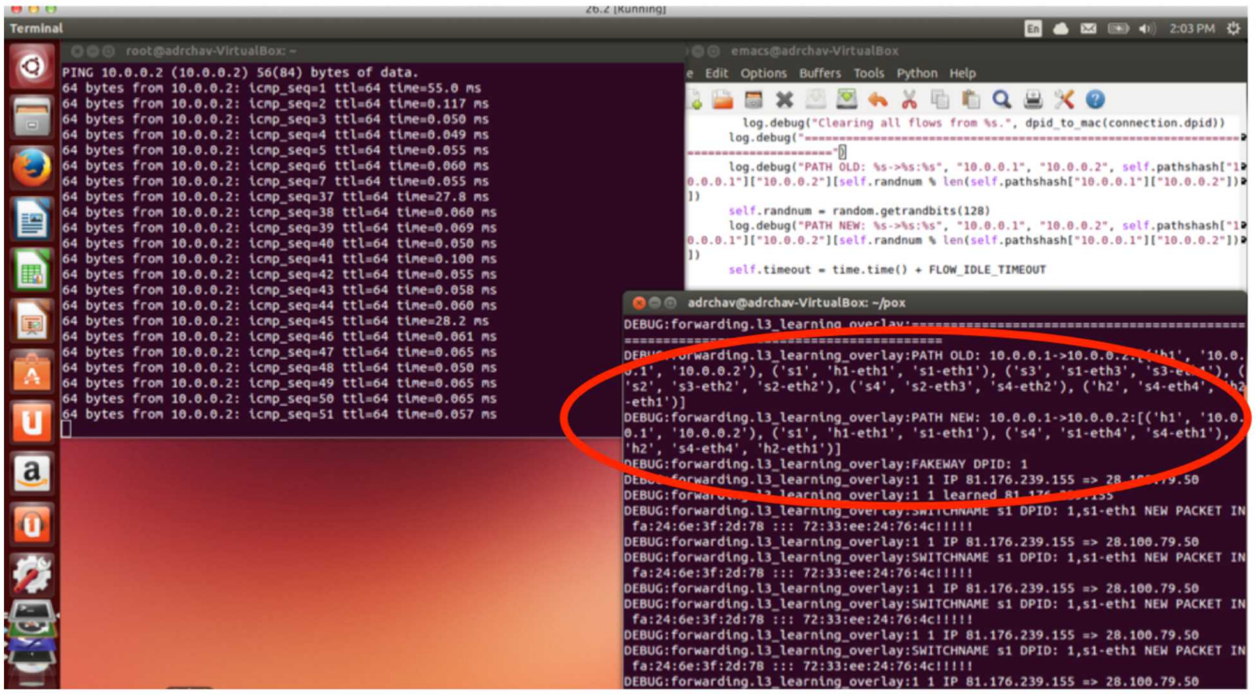
The *FindAllPaths* function uses the breadth-first-search algorithm and maintains a stack to enumerate all possible paths within the network. This cost is incurred on only the first time that the POX controller is started. During runtime, a user would pass in a parameter to specify the time intervals when the paths of network flow rules should be re-randomized. A user may also specify a maximum number of hops to help satisfy the constraints of the allowable delay in the network. The maximum number of hops could also tie into the QoS parameters for individual users of the system. If a critical process that must complete within a certain time bound, then a lower number of maximum hops can be configured.

```
// Generate every non-looping network path between each pair of systems
// within the network
GenertePaths() {
    // convert the topology into a graph
    $g = CreateGraph()
    // global hash table of paths from each pair of systems
    $pathhash = {}{}

    // enumerate and store each path between each pair of systems
    for $host1 in g:
        for $host2 in g:
            $paths = FindAllPaths($g, $host1, $host2, [])
    for $path in $paths:
        $pathshash[$host1][$host2].append($path)
}

// When the packet is received at each switch, install the flow rule that
// corresponds to the random path currently selected (which changes
// in time)
PacketIn($packet) {
    InstallRandomPathFlow($pathshash[$packet.src][$packet.dst])
    SendPacket($packet)
}
```

As an example, an ongoing session is shown in Figure 4 where packets from one host, h1, sent to another host, h2, originally take the path h1→ s1 → s2 → s3 → s4 → h2. The network topology, in this example, consists of two hosts, h1 and h2, along with four switches between the two hosts. The hosts are connected in a mesh network and the path randomization algorithms select a random path every 1 second interval of time. After a given period of time supplied by the user passes, the new path taken is re-randomized to h1 → s1 → s4 → h2.



```
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=55.0 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.117 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.050 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.060 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: icmp_seq=37 ttl=64 time=27.8 ms
64 bytes from 10.0.0.2: icmp_seq=38 ttl=64 time=0.060 ms
64 bytes from 10.0.0.2: icmp_seq=39 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=40 ttl=64 time=0.050 ms
64 bytes from 10.0.0.2: icmp_seq=41 ttl=64 time=0.100 ms
64 bytes from 10.0.0.2: icmp_seq=42 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: icmp_seq=43 ttl=64 time=0.058 ms
64 bytes from 10.0.0.2: icmp_seq=44 ttl=64 time=0.060 ms
64 bytes from 10.0.0.2: icmp_seq=45 ttl=64 time=28.2 ms
64 bytes from 10.0.0.2: icmp_seq=46 ttl=64 time=0.061 ms
64 bytes from 10.0.0.2: icmp_seq=47 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=48 ttl=64 time=0.050 ms
64 bytes from 10.0.0.2: icmp_seq=49 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=50 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=51 ttl=64 time=0.057 ms
```

```
        log.debug("Clearing all flows from %s.", dpid_to_mac(connection.dpid))
      log.debug("==============================================================
====================")
      log.debug("PATH OLD: %s->%s:%s", "10.0.0.1", "10.0.0.2", self.pathshash["1
0.0.0.1"]["10.0.0.2"][self.randnum % len(self.pathshash["10.0.0.1"]["10.0.0.2"])
])
      self.randnum = random.getrandbits(128)
      log.debug("PATH NEW: %s->%s:%s", "10.0.0.1", "10.0.0.2", self.pathshash["1
0.0.0.1"]["10.0.0.2"][self.randnum % len(self.pathshash["10.0.0.1"]["10.0.0.2"])
])
      self.timeout = time.time() + FLOW_IDLE_TIMEOUT
```

```
DEBUG:forwarding.l3_learning_overlay:============================================
========================================
DEB[illegible]forwarding.l3_learning_overlay:PATH OLD: 10.0.0.1->10.0.0.2:[('h1', '10.0.
0.1', '10.0.0.2'), ('s1', 'h1-eth1', 's1-eth1'), ('s3', 's1-eth3', 's3-e[illegible]'), (
's2', 's3-eth2', 's2-eth2'), ('s4', 's2-eth3', 's4-eth2'), ('h2', 's4-eth4', 'h2
-eth1')]
DEBUG:forwarding.l3_learning_overlay:PATH NEW: 10.0.0.1->10.0.0.2:[('h1', '10.0.
0.1', '10.0.0.2'), ('s1', 'h1-eth1', 's1-eth1'), ('s4', 's1-eth4', 's4-eth1'), (
'h2', 's4-eth4', 'h2-eth1')]
DEBUG:forwarding.l3_learning_overlay:FAKEWAY DPID: 1
DEB[illegible]forwarding.l3_learning_overlay:1 1 IP 81.176.239.155 => 28.1[illegible].79.50
DEBUG:forwa[illegible]learning_overlay:1 1 learned 81.17[illegible].155
DEBUG:forwarding.l3_learning_overlay:SWITCHNAME s1 DPID: 1,s1-eth1 NEW PACKET IN
 fa:24:6e:3f:2d:78 ::: 72:33:ee:24:76:4c!!!!!
DEBUG:forwarding.l3_learning_overlay:1 1 IP 81.176.239.155 => 28.100.79.50
DEBUG:forwarding.l3_learning_overlay:SWITCHNAME s1 DPID: 1,s1-eth1 NEW PACKET IN
 fa:24:6e:3f:2d:78 ::: 72:33:ee:24:76:4c!!!!!
DEBUG:forwarding.l3_learning_overlay:1 1 IP 81.176.239.155 => 28.100.79.50
DEBUG:forwarding.l3_learning_overlay:SWITCHNAME s1 DPID: 1,s1-eth1 NEW PACKET IN
 fa:24:6e:3f:2d:78 ::: 72:33:ee:24:76:4c!!!!!
DEBUG:forwarding.l3_learning_overlay:1 1 IP 81.176.239.155 => 28.100.79.50
DEBUG:forwarding.l3_learning_overlay:SWITCHNAME s1 DPID: 1,s1-eth1 NEW PACKET IN
 fa:24:6e:3f:2d:78 ::: 72:33:ee:24:76:4c!!!!!
DEBUG:forwarding.l3_learning_overlay:1 1 IP 81.176.239.155 => 28.100.79.50
```

**Figure 4.** Two hosts communicating through random paths; original path h1→ s1 → s2 → s3 → s4 → h2, random path taken is h1 → s1 → s4 → h2.

# 6. EXPERIMENTATION

The final step of the project was to validate that the dynamic defense and moving target defense strategies can be applied to a representative control system environment. Our team worked with project partners to develop a scenario where the ADDSec technologies could be integrated in a physical environment. The environment consisted of nine buildings with a fully functioning SDN network between the buildings. The infrastructure is being developed to harness a microgrid system; our tests were performed before end devices were introduced into the network to ensure our technology could be successfully integrated. Figure 5 shows the configuration of the network where ADDSec was integrated. Our testing involved three buildings where we were able to successfully demonstrate the ADDSec technologies. Prior to introducing the ADDSec technologies, each of the three buildings included an SDN capable switch that were preconfigured with flows to facilitate communications. The three SDN capable switches are shown in Figure 5 labeled as "SDN Switch Flow Forwarding" in the lighter blue rectangles in each of the buildings.

To introduce the ADDSec technologies, four additional systems were introduced into Building #1. The first system was an SDN capable switch, labeled "SDN Switch Flow Rewrites", that implements the optional `set_field` operations within the OpenFlow 1.3 specification. The `set_field` operations are needed so that the IP addresses can be rewritten to random values when leaving the source system and then rewritten back to the original values before reaching the destination system. An end device was included in Building #1 labeled as "Host 1" to represent an end device within the network. An additional Linksys router was added to the network to support an out-of-band communication channel for the SDN OpenFlow control traffic. The dashed green lines show the out-of-band communications.

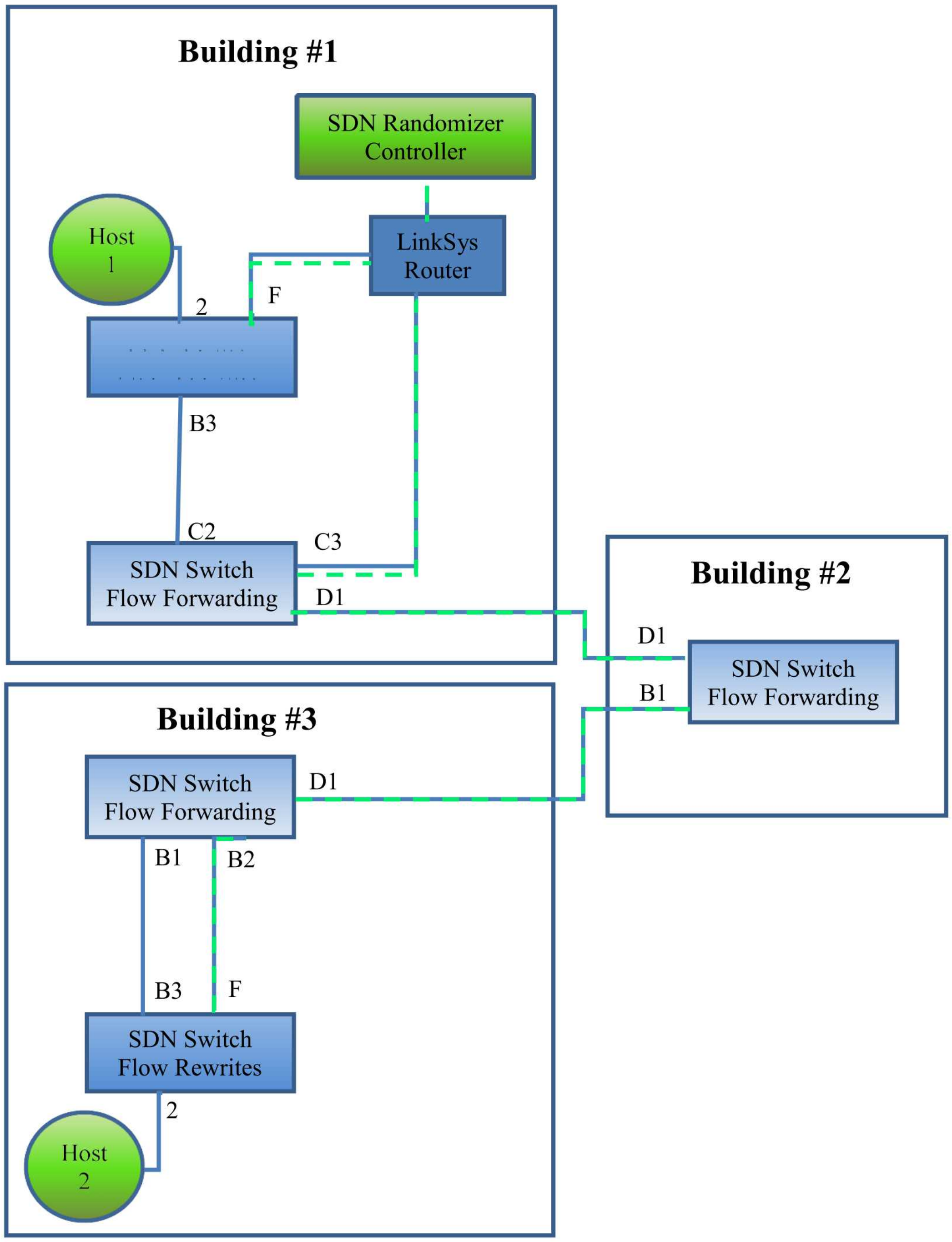


**Figure 5**. Multi-building SDN configuration with IP randomization enabled.

To manage the randomized flows that will be installed on the SDN switches, the "SDN Randomizer Controller" was also added to the network. This system is responsible for periodically communicating the randomized IP addresses to each of the SDN switches to create the moving target defense solution. The same "SDN Switch Flow Rewrites" SDN-capable switch was also introduced into Building #3. Additionally, a second host, labeled "Host 2", was added to Building #3 to show communications between two end points within the network. Building #2 only has an SDN capable switch, labeled "SDN Switch Flow Forwarding", that has flows configured to forward traffic, both control and data, between Building #1 and Building #3.

"Host 1" and "Host 2" are then able to communicate with one another with the IP randomization enabled. Additionally, "Host 1" and "Host 2" both have the machine learning dynamic defense algorithms enabled to detect anomalous behavior on the network. Flows with the source IP address of the "SDN Randomizer Controller" are identified as management communications and are forwarded to the correct ports of the "SDN Switch Flow Rewrites" which are both port "F" in the diagram.

## 6.1. Adversarial Scenario

To capture data for metrics on the feasibility of applying the ADDSec technologies to detect and respond to a potential threat, a scenario was developed to show how ADDSec could be applied. We showed that if an adversary were to be introduced into the network either as (1) insider or as (2) a result of a successfully launched man-in-the-middle attack anywhere between the two "SDN Switch Flow Rewrites", that we could detect a network scan and automatically respond by randomizing the IP addresses to invalidate the information gained from the network scan. In our specific scenario, the adversary was introduced in building #1 in the link with "B3" and "C2" as the end point ports. The adversary then launched the following nmap scan:

```
nmap -sP x.y.z.0/24
```

where x.y.z is the network address that was configured in the partner system. The network in this case was configured as a 24 bit network with the last 8-bits reserved for host IP addresses. The nmap command above will scan the entire IP space of the x.y.z network and report back the hosts that are active. In our scenario, once the adversary receives the results they would then attempt to open a secure shell (ssh) session with the hosts in the network. The goal of the ADDSec technology is to detect the initial network scan and randomize IP addresses so that the information gained about active hosts within the network would no longer be valid. To accomplish this goal, the machine learning algorithms on the end hosts would flag the scan as anomalous behavior and classify the scan as a reconnaissance attack. Based on the classification of the machine learning algorithm, an appropriate response would be generated. In this case, randomizing IP addresses would be an appropriate response to mitigate the adversary from progressing in their goals to act on the hosts that were discovered from the network scan. The described use case was successfully deployed within our partners' testbed and demonstrated to detect and respond to the network scan by randomizing IP addresses.

## 6.2. Metrics

Our team leveraged metrics to quantify the impacts to the testbed when deploying the various randomization schemes that were implemented. Data for the metrics were captured when the randomization schemes were performed independently and when combined with one another. The metrics used include round trip time, bandwidth, throughput, TCP retransmits, and dropped packets. The randomization schemes evaluated include a baseline measurement without any randomization schemes enabled, IP randomization with a frequency interval of 3 seconds, port randomization with a frequency interval of 3 seconds, and port randomization with a frequency interval of 1 second with encryption. The results are shown in Figure 6-Figure 11.

Figure 6 and Figure 7 show the results of the ICMP round trip times measured using the OpenDaylight controller as applied towards the scenario shown in Figure 5 where “Host 1” is issuing the following command to “Host 2”:

```
ping -c 300 host2.
```

In Figure 6, each of the randomization schemes are measured independently and when combined with one another. The impacts vary slightly and are within the noise of network traffic as each scheme fluctuates outperforming and underperforming the other schemes depending on when the measurement is taken. This can be seen more clearly in Figure 7 where the raw averages and standard deviations closely resemble one another across each of the randomization schemes. The impacts of each of the randomization schemes in our test environment proved to be minimal from our experimental results obtained.

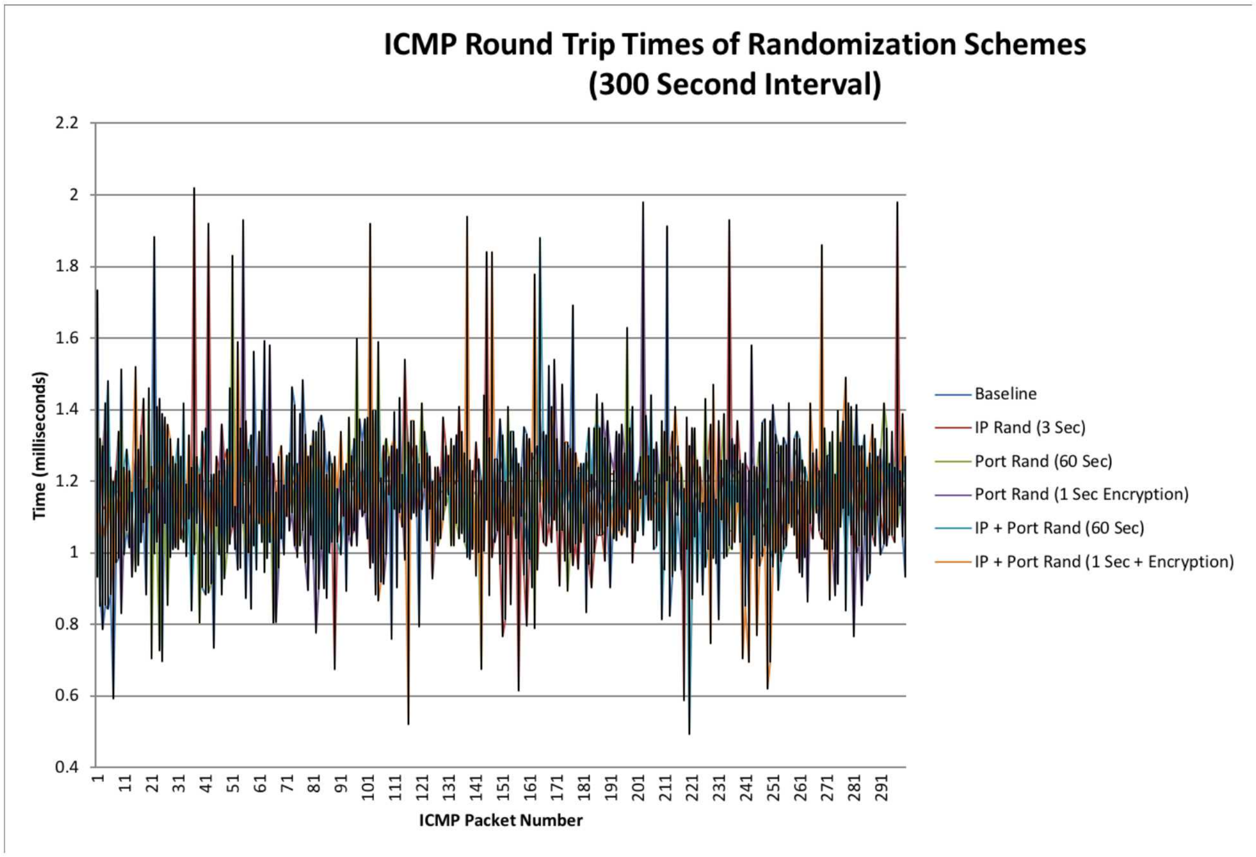
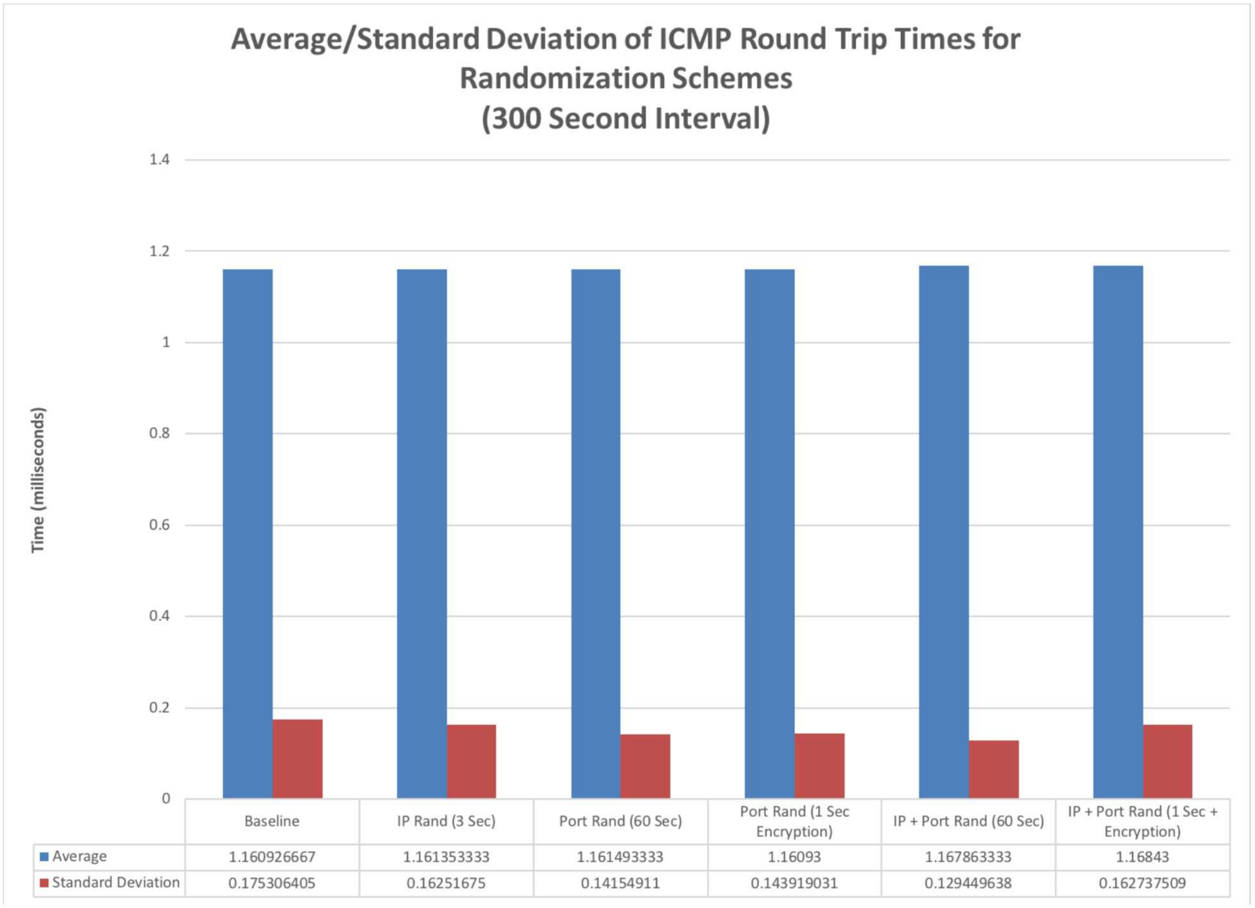


**Figure 6.** ICMP Round Trip Time measurements when enabling each randomization scheme independently and when combined with one another over a 300 second interval.



| | Baseline | IP Rand (3 Sec) | Port Rand (60 Sec) | Port Rand (1 Sec Encryption) | IP + Port Rand (60 Sec) | IP + Port Rand (1 Sec + Encryption) |
|---|---|---|---|---|---|---|
| 🟦 Average | 1.160926667 | 1.161353333 | 1.161493333 | 1.16093 | 1.167863333 | 1.16843 |
| 🟥 Standard Deviation | 0.175306405 | 0.16251675 | 0.14154911 | 0.143919031 | 0.129449638 | 0.162737509 |

**Figure 7.** The averages and standard deviation for the round trip times when enabling each of the randomization schemes independently and in combination of each other over a 300 second interval of time.

Figure 8 and Figure 9 measure the effects that each of the randomization schemes have on the transfer rates and the bandwidth. The results and other metrics captured in this section are based on the network schematic shown in Figure 5. Since we were working with 100 megabits per second links, the maximum amount of data that can be transferred is 12.5 megabytes (100/8). Our results show that most of the schemes, including the baseline, achieve ~11.2 megabytes of data transferred. The exceptions to this are the schemes that use the port randomization with encryption involved where ports are randomized every 1 second. Since we are randomizing at each of the endpoints in software and the cost for AES encryption is significant, the transfer rates were significantly impacted and yielded ~0.1 megabytes of data transferred. These results indicate that in environments where large amounts of data need to be transferred quickly, that encrypting and randomizing ports every 1 second may not be a suitable option. However, the impacts on round-trip time were minimal so if small amounts of data are transferred, as is typically the case within control system communications, then any of the schemes may be appropriate. The metrics captured for transfer rates and bandwidth were captured with the following command:

```
    iperf3 -c host2 -i 1 -t 300 -p 999 -V.
```

This command was issued for the client to connect to "Host 2" on port 99 and report back results every 1 second over a 300 second total interval in verbose mode.

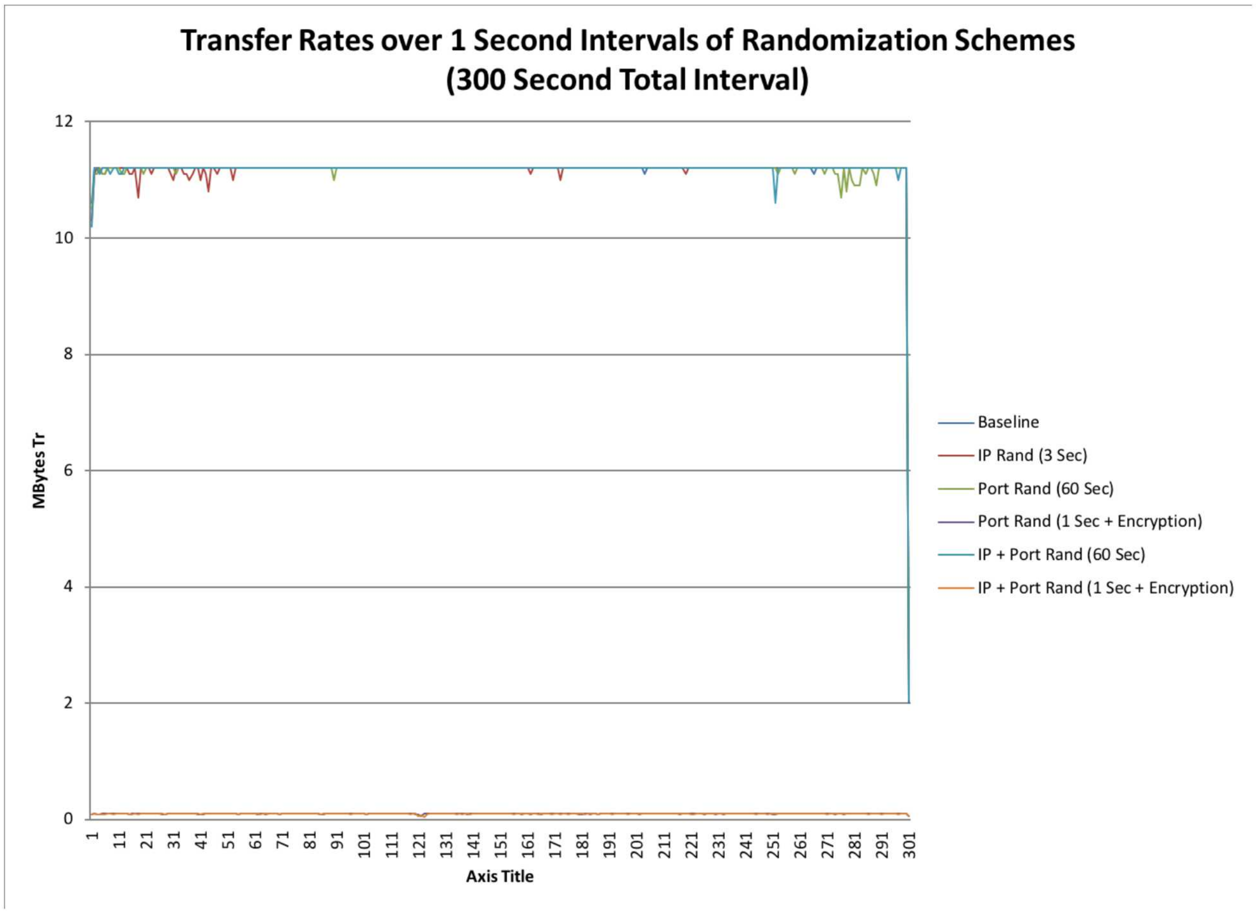


**Figure 8.** The amount of data transferred when enabling each randomization scheme independently and in combination of each other over 1 second increments in a 300 second interval of time.

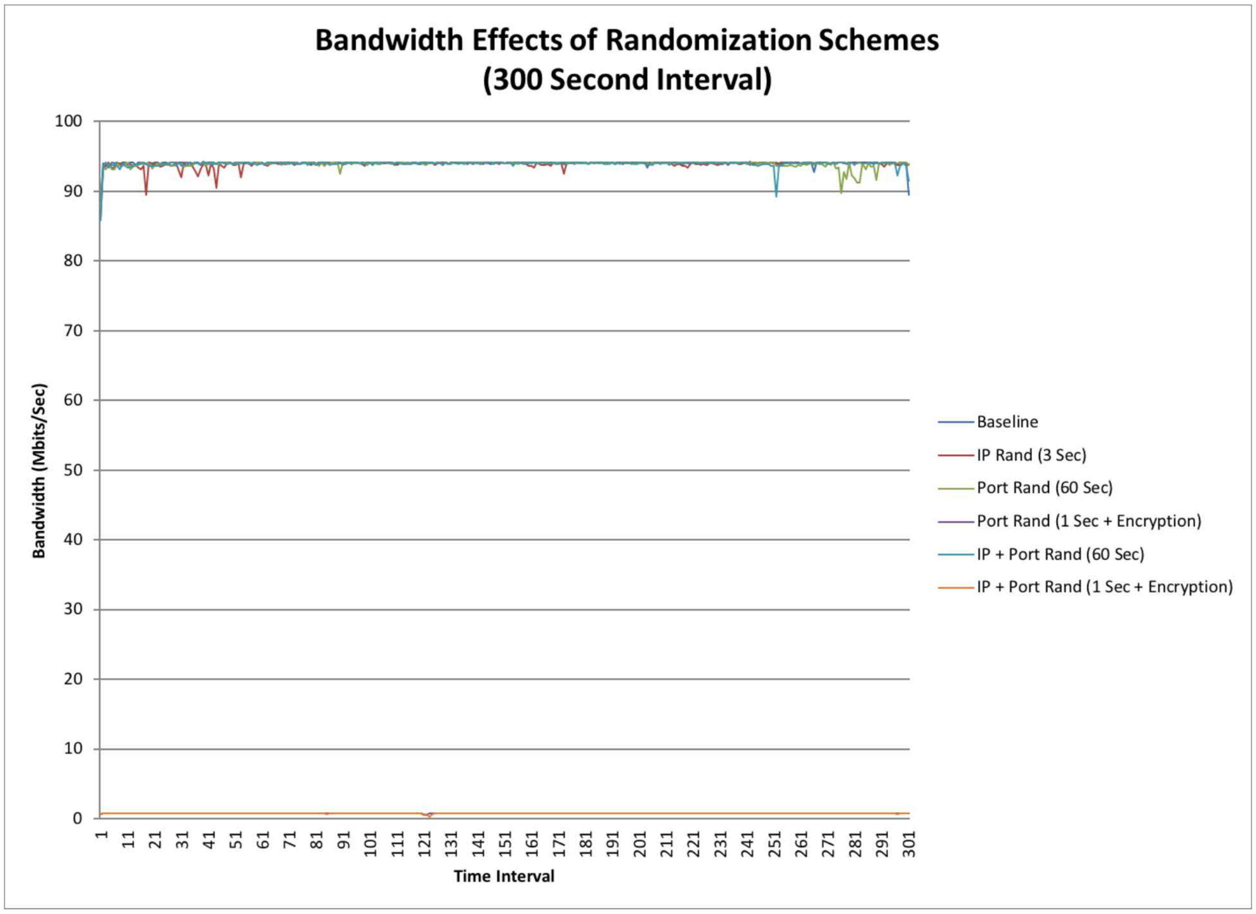


**Figure 9.** The measured bandwidth when enabling each randomization scheme independently and in combination of each other over 1 second increments in a 300 second interval of time.

Figure 10 and Figure 11 show the results of the number of retransmits incurred when each of the randomization schemes were enabled independently and in combination with one another. When measuring the baseline configuration without any of the randomization schemes enabled, 124 retransmits were recorded, or ~0.005% of the total number of packets. When enabling the IP randomization scheme and the port randomization every 60 seconds scheme independently, there were fewer retransmits and a fewer percentage of overall retransmits. This may be a result of a more congested network during the time that the baseline measurements were recorded. It would be expected that the baseline would be similar or better than when enabling each of the randomization schemes. The remaining three schemes all produced a higher percentage of retransmits, although only two had a higher number of total retransmits. This is a result of the fact that fewer packets were able to be transmitted when encrypting port numbers every one second. Although the percentages in all cases are low, it should be taken into consideration if these percentages are acceptable to be applied within an operational setting. The results obtained were captured using the same iperf command that was specified for the bandwidth and throughput measurements shown in Figure 8 and Figure 9. The number of dropped packets were also measured as part of these tests. All of the schemes reported no dropped packets.

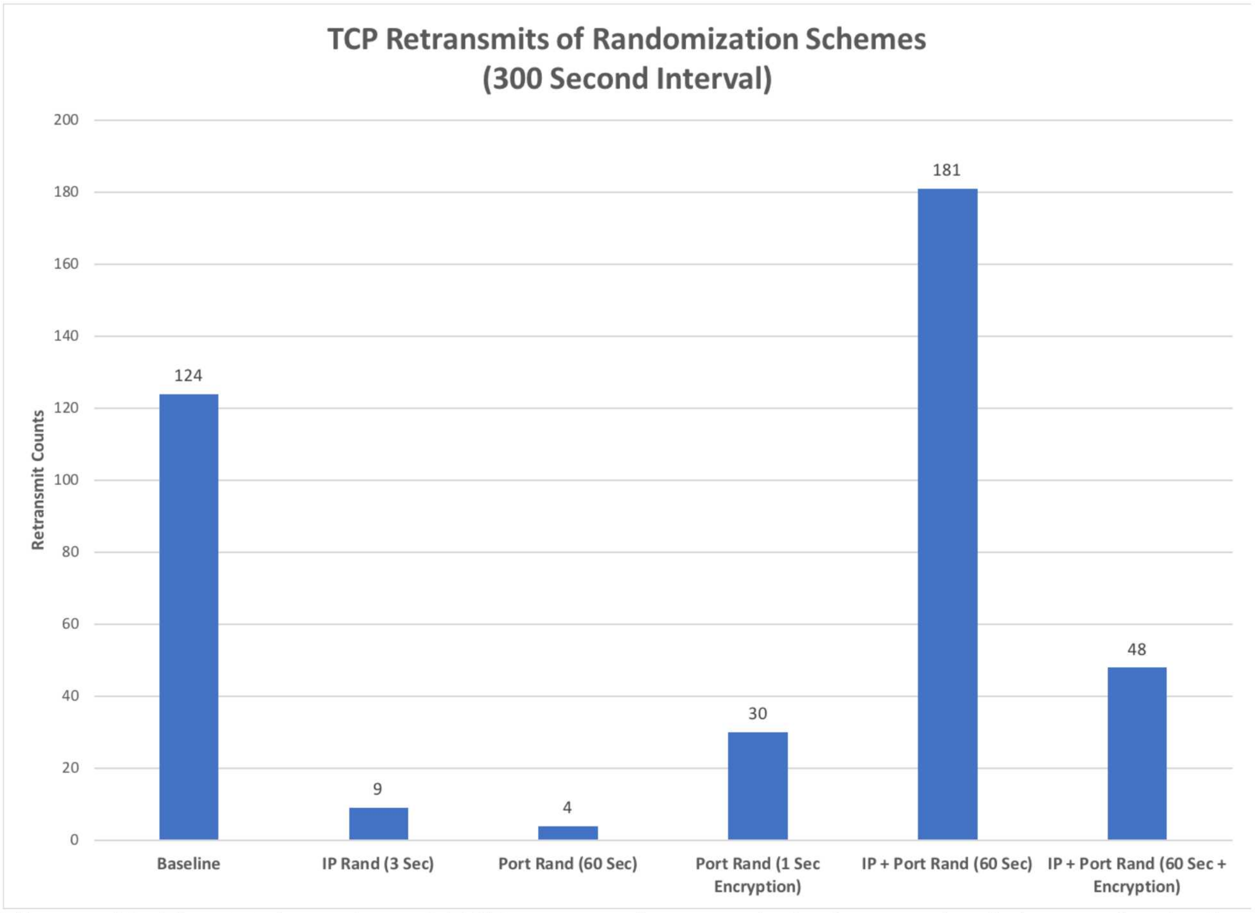


**Figure 10.** The total number of TCP retransmits recorded when each of the randomization schemes were enabled independently and in combination of each other over a 300 second interval of time.

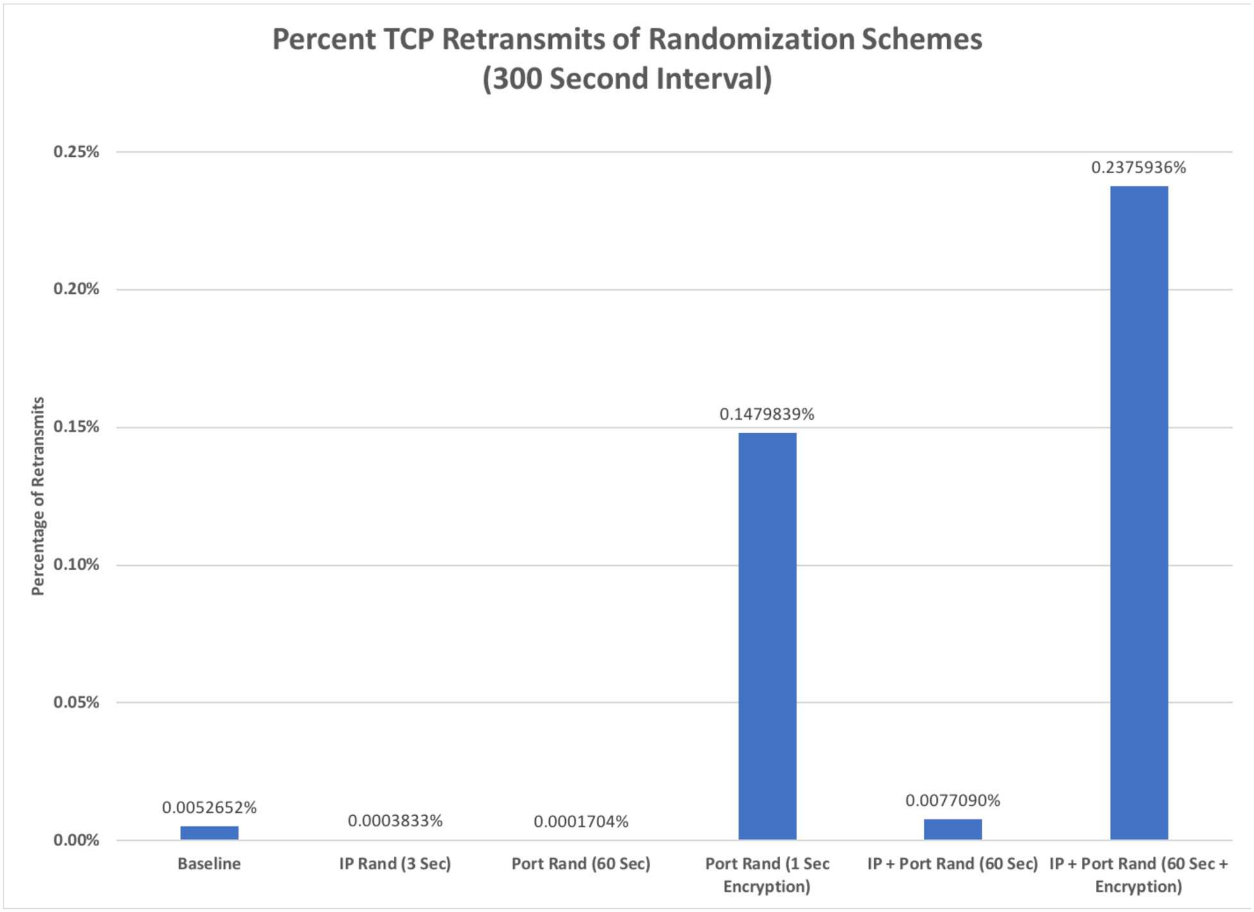


**Figure 11.** The percentage of TCP retransmits recorded when each of the randomization schemes were enabled independently and in combination of each other over a 300 second interval of time.

## 7. CONCLUSIONS

The MTD approaches we developed have been shown to be effective within an ICS environment. We performed several experiments with a variety of configurations for each MTD technique. The techniques presented, although effective individually, are meant to be a piece of the larger computer security puzzle. These MTD techniques can be thought of as additional layers of defense to help protect a system from an adversary attempting to gain an understanding of a system in the early stages of an attack. Additional defenses can be deployed alongside the MTD techniques to create an even more secure system. Deploying an individual MTD technique or a suite of MTD techniques alongside other computer security protections will depend on the application. For example, the MTD techniques may provide a mitigation to a "hitlist" type of attack [69], but the MTD techniques themselves do not provide the ability to detect the hitlist attack. Intrusion detection systems (IDSs), firewalls, security information and event management systems, and virus scanners, for example, should all be included as part of the overall security protection. In this scenario, we used machine learning algorithms to detect the hitlist attack and trigger the MTD schemes. The MTD strategies by themselves are not meant to be a comprehensive security solution that protects against all threats, but rather should be applied as an additional layer of defense in general.

### 7.1. Lessons Learned

If there are commercially available products that are under consideration for integration as part of any security solution, they should be fully evaluated to ensure that they will meet all of the requirements of the targeted use case. If there is a mismatch in performance or functionality, early identification is important. We evaluated several switches that are SDN capable and their implementations varied significantly. The variations in implementation resulted in drastically different results. The differences did not impact the feasibility of applying the MTD approaches to an ICS environment in this case, but could make a difference in environments outside of ICS. Comparing performance in the software implementation to the hardware implementation required several experiments to identify and narrow down the observed slowdown in latency to be attributed to the software based implementation.

Following on this concept, a full analysis of several software options that are candidates for integration into a system should be also performed. We leveraged several open source software packages to support the MTD techniques developed. Open source software can be an extremely valuable and cost effective solution, but these solutions are not necessarily always stable production-ready solutions. The Open Daylight controller is a large code base written in the Java programming language with numerous contributors and a large amount of features available. The complexity of Open Daylight makes it difficult to troubleshoot and correct any errors encountered. Care should be taken when selecting open source software as bugs that are discovered along the way may become problematic to correct or fit within the timeline of the project. We encountered a few OpenDaylight bugs, but we developed workarounds for those bugs. One problem we observed was when submitting a bulk amount of flows to be installed that was contained within a single transaction using a module named "bulk-o-matic." When the Open Daylight controller received the bulk flow inquiries, the bulk-o-matic module would throw several null pointer exceptions frequently but not every time, even when resubmitting the same query. The uncertainty of knowing if a flow would successfully install or fail is not an option within ICS environments. The workaround was to continuously submit queries until the bulk-o-matic module eventually accepted the bulk flow query. This solution was not ideal, but it did work after a few attempted queries so it was not a major focus of concern. This could be a major concern in a production environment because of the uncertainty about knowing when the bulk-o-matic module would actually accept and install the bulk flow.

Another lesson learned came from the experimentation process itself where we repeatedly performed experiments, each within a higher fidelity environment. Our research took the approach of running experiments in a simulated environment, then in a virtualized environment, and finally in a representative environment. The reason for this was that the simulated environment provided a means to collect results quickly to evaluate the feasibility of each MTD approach and further refine algorithms and new considerations. The virtualized environment provided a higher fidelity model that could be used to collect metrics of each MTD approach with network latency built in. Finally, the representative environment provided a platform to obtain results that did not potentially overlook pieces of the system that could not be anticipated in a simulated or virtualized environment. The sequence of tests provided insights into the effectiveness of each approach and allowed for cross-validation of each environment. The approach also allowed for easier troubleshooting early in the process before reaching the representative environment. Not all research projects will have access to a representative testbed, but in either case it is important to perform experimentation and simulations beforehand to ensure that the technology is feasible for the target environment before investing a significant amount of time and money into an approach that has not been fully tested and evaluated.

A limitation of the approaches we have presented are that they each depend on IP routable communications to operate. The concepts can be applied towards communications at layer 2 and below but have not been adapted towards those use cases at this point. The ability to introduce SDN into ICS environments may also not be an option as some environments may already be fully deployed and accredited, so it may be difficult to integrate new technologies. In this scenario, it may be difficult to convince decision makers to upgrade hardware or introduce software to support the SDN capabilities when the operational network has not experienced any catastrophic failures or cyber-attacks to date. This could also be a difficult proposition since standards and requirements must be met before introducing any technology into an ICS setting. There is also a long testing process that must be satisfied before any technology can make it to a field deployment, particularly in an ICS setting.

## 7.2. Future Work

Future areas of research can focus on expanding our research beyond ICS environments. Additional environments for potential research include cloud computing, Internet of Things (IoT), and mobile environments, each of which has their own unique sets of constraints and requirements. The tradeoffs between security and usability can similarly be analyzed and evaluated in each of those respective environments. Cloud computing has a large capacity, in terms of resources, however complete control of the physical infrastructure is lost. The loss of physical control may make some MTD defenses more difficult to deploy such as the MTD defenses that depend on specific hardware that must be installed. IoT environments may be difficult to scale path randomization since the enumeration of all possible paths in the network grows exponentially as the size of the network grows. Mobile environments may make IP randomization more challenging due to the large number of devices that are dynamically entering and leaving the network, or dealing with device handoff between base stations.

The ability to perform forensic analysis as the MTD techniques are operating is another area of future research. If the IDS's are alarming on packets and logging randomized IP addresses, that information by itself is of little value to a forensic analyst. The forensic analysis has to be tied into and aware of the MTD techniques deployed to be effective, which needs further investigation. Forensic capabilities built into the MTD techniques will be important for analysis in the event of a system compromise or also when actively observing the current state of the network. For better situational awareness, the MTD techniques should be available for security operations personnel to understand the current state of the network.

Autonomous systems are another area of future research for each of the MTD techniques discussed. If the SDN controller fails or one of the SDN capable switches fail, instabilities within the network will occur. To address this problem, building autonomy into the design is a possible solution. With autonomy built-in, the SDN controllers and SDN switches can continue to operate and self-heal after a failure occurs. Currently, fault tolerant algorithms are developed to protect the SDN controller from failures. However, if all controllers fail, an interesting area of research is whether or not the switches can detect the controller failures and continue to operate on their own and take over the role of the controller. Similarly, if any of the SDN switches fail, an additional area of research would be to detect such failures and automate the reconfiguration of the network parameters to continue to operate as expected.

Another area for future research is to investigate the MTD techniques being applied and retrofitted into environments where SDN is not prevalent. In the case of traditional networks, an end point solution may be an appropriate area to apply the MTD techniques. As an endpoint solution, however, the MTD techniques should be designed so that network connections are not broken as network configurations are randomized. For IP randomization, modifications to the kernel would be necessary to maintain established connections within the TCP/IP stack during reconfiguration periods. Another solution outside of the endpoint solution would be to introduce the MTD as a gateway device. This was the approach taken as part of our research. A related potential area for future research is to borrow techniques from metamorphic code generation typically used by an adversary, but instead apply those strategies as a computer defense. Continuously modifying software so that it is functionally equivalent but continuously changing implementations in time would increase the difficulty for an adversary to exploit software vulnerabilities.

An additional area of interest to our research is the amount of information that can be gained from side channel attacks on the proposed MTD techniques. Specifically, of interest to our research is the information that can be gained from an adversary who obtains knowledge about the parameters of the MTD techniques and what advantages that provides them. One example of a side-channel attack is an adversary learning the frequencies at which the MTD strategies reconfigure the system based on network latency measurements. In such a scenario, investigating possible mitigations to defend against such side-channel based attacks would be of interest.

## 7.3. Summary

We have developed several MTD approaches with the goal of introducing additional security protections to ICS environments. ICS networks are typically statically configured and have predictable communication patterns that do not change over extended periods of time. We have developed IPv4 randomization, application port number randomization, and network communication path randomization to serve as MTD strategies for ICS networks. We evaluated all of the strategies for their effectiveness within an ICS environment. We implemented the MTD solutions as an SDN based solution and as an endpoint based solution. The SDN solution had the benefit of being a scalable solution that was transparent to the end devices, whereas the end point solution had the benefit of building the security protections directly into the end devices.

Each of the MTD strategies proved to be feasible within an ICS setting, as each of them increased latency by less than 0.01 ms on average. The environments evaluated as part of our research included a simulated environment, a virtualized environment and a representative environment. The simulated environment was a standalone system using local processes to simulate an adversary and a defender. The virtualized environment consisted of virtual machines to model an ICS system with the same ICS protocols enabled that would typically be seen in a field deployment. The representative environments included physical systems that harnessed the defender systems and the adversary systems.

The operational measurements captured included bandwidth, throughput, TCP retransmits, and number of dropped packets. Although bandwidth and throughput are typically not critical in most ICS environments due to the low bandwidth communications, the evaluation was performed in the event of synchrophasor measurements being a part of the network communications, which can consume bandwidth rates of 150 Mbits/sec [70].

We evaluated several SDN capable switches, including two hardware switches and one software switch. The two hardware switches varied in their functionality. The first switch performed the match criteria and the action for the flows within software. The second switch performed these same operations in hardware. The latency was drastically reduced in the hardware implementation compared to the software implementation. The delays of the hardware switch were on the order of microseconds, while the software flow implementation gave results that were delayed on the order of milliseconds. The open source software switch evaluated also gave results that were in the milliseconds of increased delay. All SDN capable switches were well within the constraints of a typical ICS environment, but the hardware switch had the best performance results observed.

The MTD approaches presented here are meant to provide an additional layer of defense to an ICS system. The MTD approaches are not intended to be deployed by themselves, but rather integrated into a system of security tools to achieve an elevated overall level of security. A number of open problems that can build upon our research include evaluating these approaches in environments outside of ICS, exploring possible options of developing the same techniques for systems that have not adopted SDN, integrating the MTD approaches within existing security monitoring systems, developing the MTD solutions so that they can run autonomously if needed, and introducing diversity into the SDN controllers and SDN switches deployed. Each of the future areas of research can improve and enhance the existing concepts to create more resilient solutions that complement and build upon our research. We have shown that MTD techniques can be successful in an ICS setting and they have the potential of being applied more broadly towards other general computing environments.

## REFERENCES

1. Stouffer, K, Falco, J., Scarfone, K.. "Guide to Industrial Control Systems (ICS) Security," NIST special publication, vol. 800, no. 82, pp. 16-16, 2011.
2. Chandia, R., Gonzalez, J., Kilpatrick, T., Papa, M., and Shenoi, S. "Security Strategies for SCADA Networks," in Critical Infrastructure Protection, pp. 117–131, Springer, 2007.
3. Cardenas, A., Amin, S., Lin, Z., Huang, Y., Huang, C., and Sastry, S. "Attacks Against Process Control Systems: Risk Assessment, Detection, and Response," in Proceedings of the 6th ACM symposium on information, computer and communications security, pp. 355–366, ACM, 2011.
4. Huang, Y., Cardenas, A., Amin, S., Lin, Z., Tsai, H., and Sastry, S. "Understanding the Physical and Economic Consequences of Attacks on Control Systems," International Journal of Critical Infrastructure Protection, vol. 2, no. 3, pp. 73–83, 2009.
5. Miller, B., and Rowe, D. "A Survey SCADA of and Critical Infrastructure Incidents," in Proceedings of the 1st Annual conference on Research in information technology, pp. 51–56, ACM, 2012.
6. Falliere, N., Murchu, L., and Chien, E. "W32. Stuxnet Dossier," White paper, Symantec Corp., Security Response, vol. 5, 2011.
7. Pourbeik, P., Kundur, P., and Taylor, C., "The Anatomy of a Power Grid Blackout," IEEE Power and Energy Magazine, vol. 4, no. 5, pp. 22–29, 2006.
8. Liang, G., Weller, S., Zhao, J., Luo, F., and Dong, Z. "The 2015 Ukraine Blackout: Implications for False Data Injection Attacks," IEEE Transactions on Power Systems, vol. 32, pp. 3317–3318, July 2017.
9. Sridhar, S., and Govindarasu, M. "Data Integrity Attacks and Their Impacts on SCADA Control system," in IEEE PES General Meeting, pp. 1–6, July 2010.
10. Farhangi, H. "The Path of the Smart Grid," Power and energy magazine, IEEE, vol. 8, no. 1, pp. 18–28, 2010.
11. Robles, R., Choi, M., Cho, E., Kim, S., Park, G., and Lee, J. "Common Threats and Vulnerabilities of Critical Infrastructures," International journal of control and automation, vol. 1, no. 1, pp. 17–22, 2008.
12. Hauser, C., Bakken, D., and Bose, A. "A Failure to Communicate: Next Generation Communication Requirements, Technologies, and Architecture for the Electric Power Grid," IEEE Power and Energy Magazine, vol. 3, no. 2, pp. 47– 55, 2005.
13. Hofmeyr, S., and Forrest, S. "Architecture for an Artificial Immune System," Evolutionary Computation, vol. 8, no. 4, pp. 443–473, 2000.
14. Al-Shaer, E., Duan, Q., Jafarian, J. "Random Host Mutation for Moving Target Defense," in SecureComm, pp. 310–327, Springer, 2012.
15. Antonatos, S., Akritidis, P., Markatos, E., and Anagnostakis, K. "Defending Against Hitlist Worms Using Network Address Space Randomization," Computer Networks, vol. 51, no. 12, pp. 3471 – 3490, 2007.
16. Farris, K., and Cybenko, G. "Quantification of moving target cyber defenses," in SPIE Defense+ Security, pp. 94560L–94560L, International Society for Optics and Photonics, 2015.
17. Sharon, A., Levy, R., Cohen, Y., Haiut A., Stroh, A., Raz, D. "Automatic Network Traffic Analysis," Oct. 24 2000. US Patent 6,137,782.
18. Goldschlag, D., Reed, M., and Syverson, P. "Onion Routing for Anonymous and Private Internet Connections," Communications of the ACM, vol. 42, no. 2, pp. 39–41, 1999.
19. Shmatikov, V., and Wang, M. "Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses," Computer Security–ESORICS 2006, pp. 18–33, 2006.
20. Raymond, J. "Traffic Analysis: Protocols, Attacks, Design Issues and Open Problems," in Designing Privacy Enhancing Technologies, pp. 10–29, Springer, 2001.

21. Dingledine, R., Mathewson, N., and Syverson, P. “Tor: The Second-Generation Onion Router,” in Usenix Security, 2004.
22. “The Tor Project.” https://metrics.torproject.org/torperf.html, 2014.
23. Chakravarty, S., Barbera, M., Portokalidis, G., Polychronakis, M., and Keromytis, A. “On the Effectiveness of Traffic Analysis Against Anonymity Networks Using Flow Records,” in PAM, pp. 247–257, Springer, 2014.
24. Tchabe G., and Xu, Y. “Anonymous Communications: A Survey on I2P,” CDC Publication Theoretische Informatik-Kryptographie und Computeralgebra (https://www. cdc. informatik. tu-darmstadt. de), 2014.
25. Keromytis, A., Misra, V., and Rubenstein, D. “SOS: An Architecture For Mitigating DDoS Attacks,” IEEE Journal of Selected Areas in Communications, vol. 22, no. 1, pp. 176–188, 2004.
26. Ahsan, K., and Kundur, D. “Practical Data Hiding in TCP/IP,” in Proc. Workshop on Multimedia Security at ACM Multimedia, vol. 2, 2002.
27. Pimenidis, L., and Kolsch, T. “Transparent Anonymization of IP Based Network Traffic,” In Proceedings of 10th Nordic Workshop on Secure IT-Systems, 2005.
28. Song, Jungsuk, et al. "Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation." Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security. ACM, 2011.
29. The third international knowledge discovery and data mining tools competition dataset, KDD99-Cup, 1999. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.
30. Haas, Jason J., J. D. Doak, and Jason R. Hamlet. "Machine-oriented biometrics and cocooning for dynamic network defense." Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop. ACM, 2013.
31. Pyle (1999). Data Preparation for Data Mining. pp. 271–274, 355–359.
32. Bloom, Burton H. "Space/time trade-offs in hash coding with allowable errors." Communications of the ACM 13.7 (1970): 422-426.
33. Dietterich, Thomas G. "Ensemble methods in machine learning." Multiple classifier systems. Springer Berlin Heidelberg, 2000. 1-15.
34. Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20.3 (1995): 273-297.
35. Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.
36. Demšar, J., Curk, T., & Erjavec, A. Orange: Data Mining Toolbox in Python; Journal of Machine Learning Research 14(Aug):2349−2353, 2013.
37. Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." The Journal of Machine Learning Research 13.1 (2012): 281-305.
38. Matthews, Brian W. "Comparison of the predicted and observed secondary structure of T4 phage lysozyme." Biochimica et Biophysica Acta (BBA)-Protein Structure 405.2 (1975): 442-451.
39. Töscher, Andreas, Michael Jahrer, and Robert M. Bell. "The bigchaos solution to the netflix grand prize." Netflix prize documentation (2009).
40. Schapire, Robert E. "The strength of weak learnability." Machine learning 5.2 (1990): 197-227.
41. Borji, Ali. "Combining heterogeneous classifiers for network intrusion detection." Advances in Computer Science–ASIAN 2007. Computer and Network Security. Springer Berlin Heidelberg, 2007. 254-260.
42. Kishimoto, Kazuya, Hirofumi Yamaki, and Hiroki Takakura. "Improving performance of anomaly-based ids by combining multiple classifiers." Applications and the Internet (SAINT), 2011 IEEE/IPSJ 11th International Symposium on. IEEE, 2011.

43. Symons, Christopher T., and Justin M. Beaver. "Nonparametric semi-supervised learning for network intrusion detection: combining performance improvements with realistic in-situ training." Proceedings of the 5th ACM workshop on Security and artificial intelligence. ACM, 2012.
44. POX (About POX | NOXRepo) http://www.noxrepo.org/pox/about-pox/
45. Open vSwitch http://openvswitch.org/
46. OpenFlow Protocol, Open Networking Foundation https://www.opennetworking.org/sdn-resources/openflow
47. Tavallaee, Mahbod, et al. "A detailed analysis of the KDD CUP 99 data set." *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*. 2009.
48. TCPDUMP & Libpcap, http://www.tcpdump.org/
49. The Bro Network Security Monitor, https://www.bro.org/
50. strace(1) – Linux man page, http://linux.die.net/man/1/strace
51. Systat Utilities: SAR, http://sebastien.godard.pagesperso-orange.fr/man_sar.html
52. sar(1) – Linux man page, http://linux.die.net/man/1/sar
53. Hofmeyr, Steven A., Stephanie Forrest, and Anil Somayaji. "Intrusion detection using sequences of system calls." *Journal of computer security* 6.3 (1998): 151-180.
54. Shavlik, Jude. *Selection, Combination, and Evaluation of Effective Software Sensors for Detecting Abnormal Usage of Computers Running Windows NT/2000*. Shavlik Technologies White Bear Lake MN, 2002.
55. Shavlik, Jude, and Mark Shavlik. "Selection, combination, and evaluation of effective software sensors for detecting abnormal computer usage." *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004.
56. Dhanabal, L., and Dr SP Shantharajah. "A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms." *International Journal of Advanced Research in Computer and Communication Engineering* 4.6 (2015).
57. Panda, Mrutyunjaya, Ajith Abraham, and Manas Ranjan Patra. "Discriminative multinomial naive bayes for network intrusion detection." *Information Assurance and Security (IAS), 2010 Sixth International Conference on*. IEEE, 2010.
58. Tavallaee, Mahbod, et al. "A detailed analysis of the KDD CUP 99 data set." *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*. 2009.
59. Revathi, S., and A. Malathi. "A detailed analysis on NSL-KDD dataset using various machine learning techniques for intrusion detection." *International Journal of Engineering Research and Technology. ESRSA Publications* (2013).
60. Ingre, Bhupendra, and Anamika Yadav. "Performance analysis of NSL-KDD dataset using ANN." *Signal Processing And Communication Engineering Systems (SPACES), 2015 International Conference on*. IEEE, 2015.
61. Mukherjee, Saurabh, and Neelam Sharma. "Intrusion detection using naive Bayes classifier with feature reduction." *Procedia Technology* 4 (2012): 119-128.
62. Heba, F. Eid, et al. "Principle components analysis and support vector machine based intrusion detection system." *2010 10th International Conference on Intelligent Systems Design and Applications*. IEEE, 2010.
63. *Oliveira, R., Shinoda, A., Schweitzer, C., and Prete, L. "Using Mininet for Emulation and Prototyping Software-Defined Networks," in Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on, pp. 1-6, IEEE, 2014.*

*64. Medved, J., Varga, R., Tkacik, A., and Gray, K., "Opendaylight: Towards a Model-Driven SDN Controller Architecture," in 2014 IEEE 15th International Symposium on, pp. 1-6, IEEE, 2014.*
65. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69-74, 2008.
66. Rekhter, Y., Moskowitz, B., Karrenberg, D., Groot, G., and Lear, E. "Address Allocation for Private Internets," tech. rep., 1996.
67. Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V., and Smeliansky, R. "Advanced Study of SDN/OpenFlow Controllers," in Proceedings of the 9th central & eastern European software engineering conference in Russia, p. 1, ACM, 2013.
68. West, D. Introduction to Graph Theory, vol. 2. Prentice hall Upper Saddle River, 2001.
69. Staniford, S., Paxson, V., and Weaver, N. "How to Own the Internet in Your Spare Time," in USENIX Security Symposium, vol. 2, pp. 14{15, 2002.
70. Narendra, K., and Weekes, T. "Phasor Measurement Unit (PMU) Communication Experience in a Utility Environment," in Canadian National Committee on the International Council of Large Electric Systems (CIGRE) Conference on Power Systems, pp. 1C9{21, 2008.
71. Ryu SDN Framework, https://osrg.github.io/ryu/

## DISTRIBUTION

| | | | |
|---|---|---|---|
| 1 | MS0671 | Jennifer M. Depoy | 05828 (electronic copy) |
| 1 | MS0671 | Jason R. Hamlet | 05827 (electronic copy) |
| 1 | MS0671 | William A. Zortman | 05827 (electronic copy) |
| 1 | MS0672 | Adrian R. Chavez | 05829 (electronic copy) |
| 1 | MS0672 | Shawn E. Taylor | 05829 (electronic copy) |
| 1 | MS0813 | Han Wei Lin | 09315 (electronic copy) |
| 1 | MS0813 | William MS Stout | 09315 (electronic copy) |
| 1 | MS0933 | Brian Gaines | 09366 (electronic copy) |
| 1 | MS1138 | Rossitza Homan | 08815 (electronic copy) |
| 1 | MS0899 | Technical Library | 9536 (electronic copy) |
| 1 | MS0161 | Legal Technology Transfer Center | 11500 (electronic copy) |

Sandia National Laboratories