SAND2015-7973C

Exceptional service in the national interest

Sandia National Laboratories

Techniques for the Dynamic Randomization of Network Attributes

William M.S. Stout

Overview

- Introduction/Background
- Threat Model
- Implementation Details
  - Port Randomization
  - IP Randomization
  - Route Randomization
- Experiment Setup and Results
- Conclusions

Introduction

Computer networks, in particular Critical Infrastructure (CI) systems, continue to foster predictable communication paths and static configurations that provide a vector for accessing the critical assets of a network.

- Among the many vulnerabilities that CI systems possess, a static configuration can make CI systems attractive and easy targets for cyber-attack.

We provide a means to address these attack vectors by automatically reconfiguring network settings and randomizing application communications dynamically. Applying these protective measures will convert CI systems into "moving targets" that proactively defend themselves against attack.

Introduction

The goal of our research is to significantly reduce the class of adversaries able to rely on known static Internet Protocol (IP) addresses of CI network devices to launch an attack. Our approach introduces uncertainty and unpredictability for adversaries reconnoitering a network to determine the function of nodes on computer networks, and also for adversaries attempting to map the topology of computer networks. It is comprised of the following techniques:

- Network Randomization for TCP/UDP Ports
- Network Randomization for IP Addresses
- Network Randomization for Network Paths

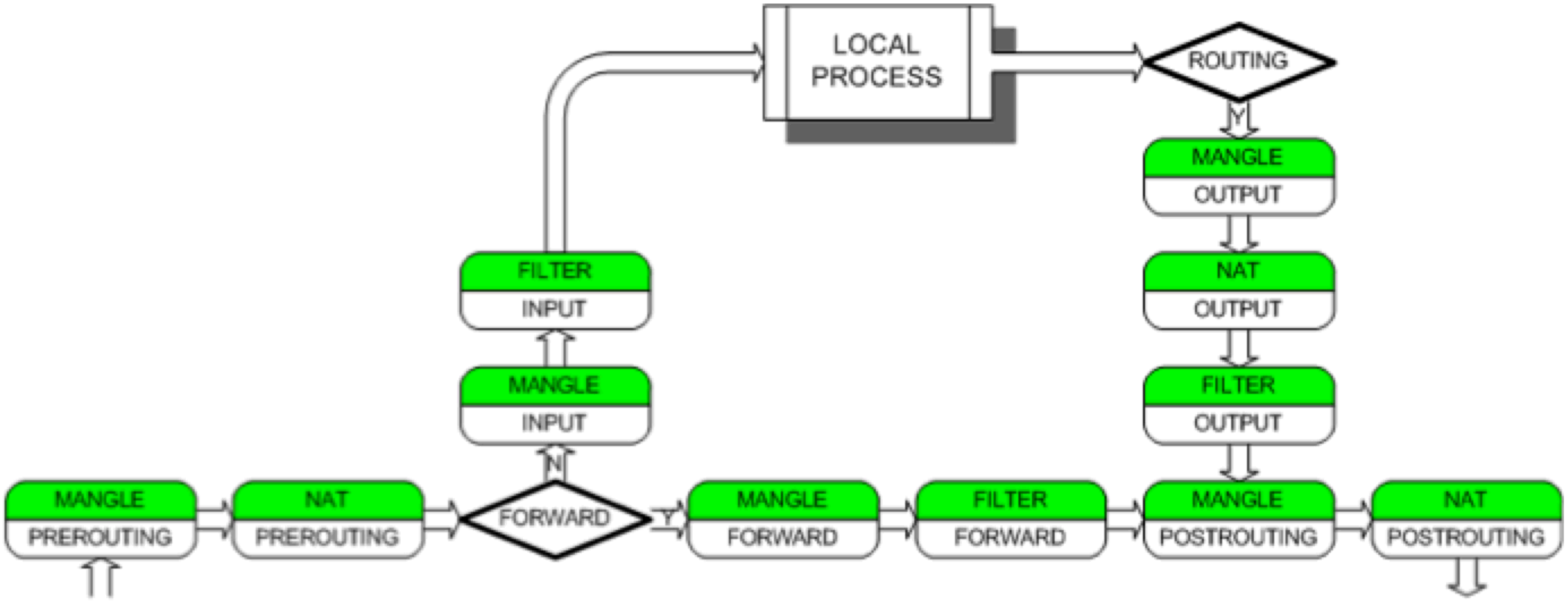
Assumption: Threat Model

- The network randomization (NR) portion of our approach assumes that an adversary has successfully gained access to a system and is able to observe traffic within the network.
- The goal of the adversary may be D/DoS, reconnaissance, targeting a specific service, or targeting a specific host on the network.
- Our goal is to prevent that adversary from learning the true IP addresses and port numbers of the services being offered on a network to mitigate the scope of damage of targeted reconnaissance attacks

Implementation: Port Hopper

- Port randomization has been implemented on each host within a network with the aid of the netfilter kernel module.
- Port randomization implementation leverages the Network Address Translation (NAT) iptable filter chain. Rules applied to the NAT chain allow a user to filter just before a packet has been routed on the incoming interface and just before a packet leaves the outgoing interface.
- The NAT filter rules allow a user to redirect or overwrite port numbers and IP addresses in each packet if it matches user-specified IP header parameters

Implementation: Port Hopper



Implementation: Port Hopper

```
// well known ports
$portMaps = [1, 2, ..., 1023]
random.seed(time())
// random permutation of well known ports
random.shuffle($portMaps)
for $i = 0 to len($portMaps) do:
   // map a random port number coming into
   // a system to a well known port number
   $inboundRule = iptables -t nat
        -A PREROUTING -p tcp
        --dport $portMaps[$i] -j DNAT
        --to-destination 127.0.0.1:($i+1)
   // map a well known port number leaving
   // a system to a random port number
  $outboundRule = iptables -t nat

       -A OUTPUT
        -p tcp --dport ($i+1) -j DNAT
        --to-destination

        127.0.0.1:$portMaps[i]

   // Apply the rules to the system

   syscall($inboundRule)

   syscall($outboundRule)
```

- All nodes synchronize clocks and seed the random number generator with time.
- the port mappings are randomized and iterated to create random mappings. The random mappings are then appended to an incoming PREROUTING *iptables* NAT rule as well as to an outgoing OUTPUT rule.
- Both rules to create and invert the mappings are necessary for both sides of the communication channel to be aware of the translation.

Implementation: IP Randomizer

- The IP address randomization application is a multi-component module written for an OpenFlow-based software-defined-networking (SDN) controller.
- At the heart of the IP address randomizer are three components:
  - the network randomization algorithm and OpenFlow interface (*nwr*)
  - the random IP address generator (*gen*)
  - the network mapping database module (*netmap*)
- Ancillary modules/code provide for a RESTful API via which an external application may trigger a force-randomization action for the network.

Implementation: IP Randomizer

NWR Module

- Listens for OpenFlow (OF) events regarding the connection of OF compatible switches.
- Uses a deny-by-default approach, switches will only switch packets when said packets match installed flow rules on the switch. If there is not a flow rule that the packet may match against, the packet header is sent to the controller to determine how to treat the packet.
- Hosts an address-resolution protocol (ARP) server.
- Only permits communication (and random IP address assignment) between entities that are part of the network as specified, and are directly connected to an OpenFlow-compatible switch.
- two implementations: (1) reactive IP address randomization, and; (2) proactive IP address randomization.
- A roll interval prescribes the timeout period for each of the randomization flow-rules.

Implementation: IP Randomizer

GEN Module

- Contains the logic for random IP address generation.
- It contains a queue data structure whose depth is initialized with the size of network (total assignable IP addresses under the defined network length). Its purpose is to keep track of the used random IP addresses, so as to avoid reassignment or collision.
- Additionally, an array is kept to track the random IP address and the true MAC address of the endpoint. This is primarily used for ARP responses to gateways that may not be part of the subnetworks under randomization.

Implementation: IP Randomizer

NETMAP Module

- Provides the necessary interface backend database that stores the true network map(s). All entries are derived from a network specification file. The *netmap* component itself consists of the several functions to aid the primary *nwr* switch:
  - getSource function is used by *nwr* to verify that a packet received from some IP is allowed to be within the network(s) under randomization. If the data is invalid, nothing is returned and the packet(s) is dropped.
  - getDest function does a similar test on the destination IP address for the packet. If the destination IP address is not in the database, nothing is returned (and the packet is dropped)

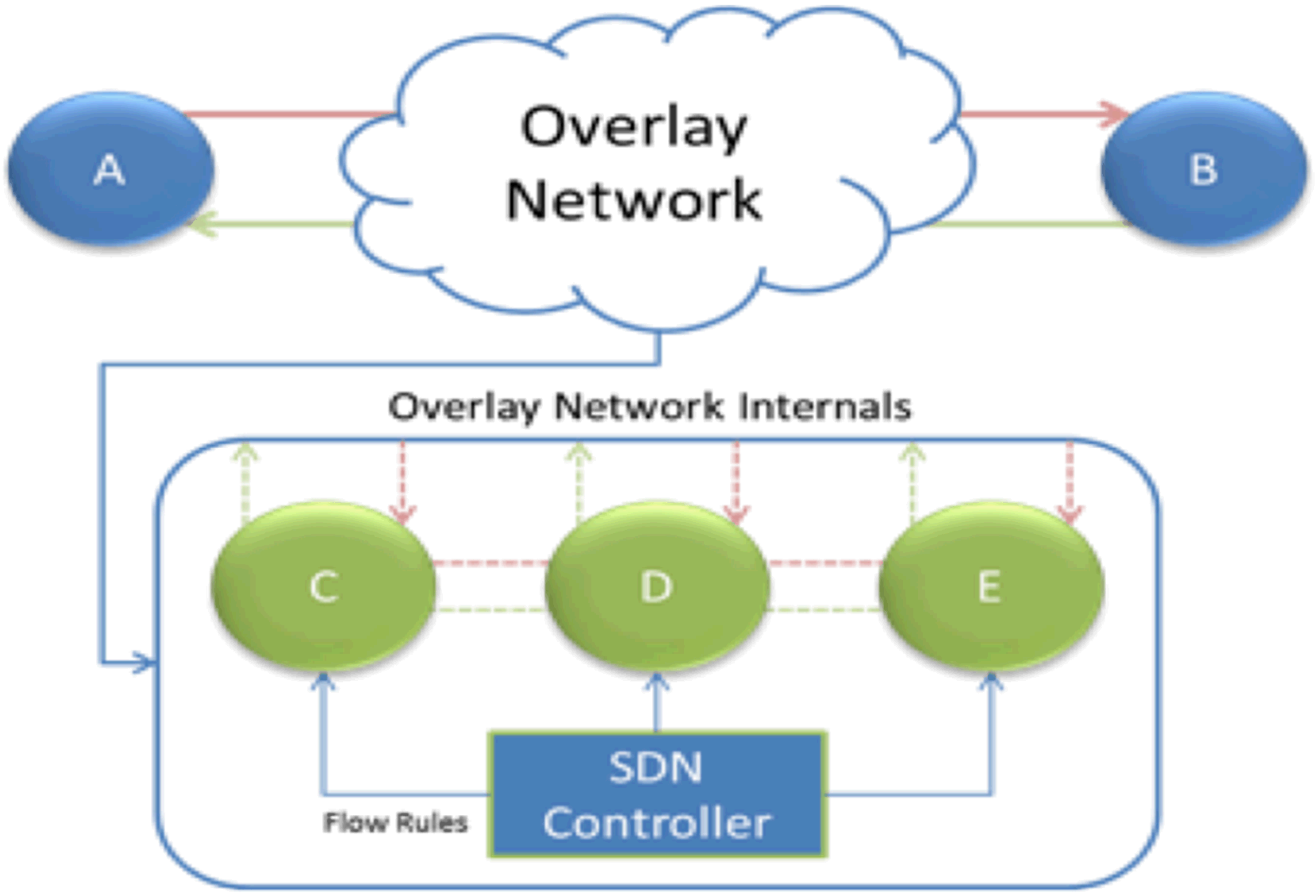
Implementation: Route Randomizer

- To randomize paths, the network topology describing endpoints and interconnections is needed to compute all possible network paths between each pair of nodes that does not contain loops.
- When two endpoints communicate, a random path is selected as the circuit switched line of communication within the overlay network for a configurable period of time. This circuit is periodically randomized and is user configurable on how frequently it is randomized.
- A Breadth First Search (BFS) algorithm is used to generate all possible paths. A stack is maintained to ensure that no paths are included that contain loops to prevent. Since the paths packets take within the network are constantly changing, an adversary must correlate traffic at every switch participating in the overlay network instead of just the entry and exit points of packets

Implementation: Route Randomizer

- Our implementation operates within an SDN setting where the controller is responsible for learning the topology and installing random flows for packets to traverse.
- The initialization costs associated with the path randomization implementation itself are the storage and computational costs to calculate all possible non-looping paths between each pair of nodes. Because the paths are constantly changing, our implementation proactively installs flows to avoid the same startup costs each time paths are randomized.
- The performance overheads vary depending on network topology, link speeds, and additional hops taken within the randomly selected network path.

Implementation: Route Randomizer



Experiment Setup

- The experimental setup consisted of a highly-instrumented testbed using virtual machines running the three different randomization techniques.
- For the port and IP hopping techniques, each endpoint in the virtual system ran network performance monitoring software to capture performance data for TCP and UDP; the former was comprised of bandwidth counts, while the latter consisted of bandwidth, jitter and packet loss counts. Included in the parameters for testing were the rotation intervals for the reactive IP address randomizer.
- To test route randomization, we used the IPerf [16] tool to measure Round Trip Times (RTT), bandwidth, and data transfer times.

Experiment Results

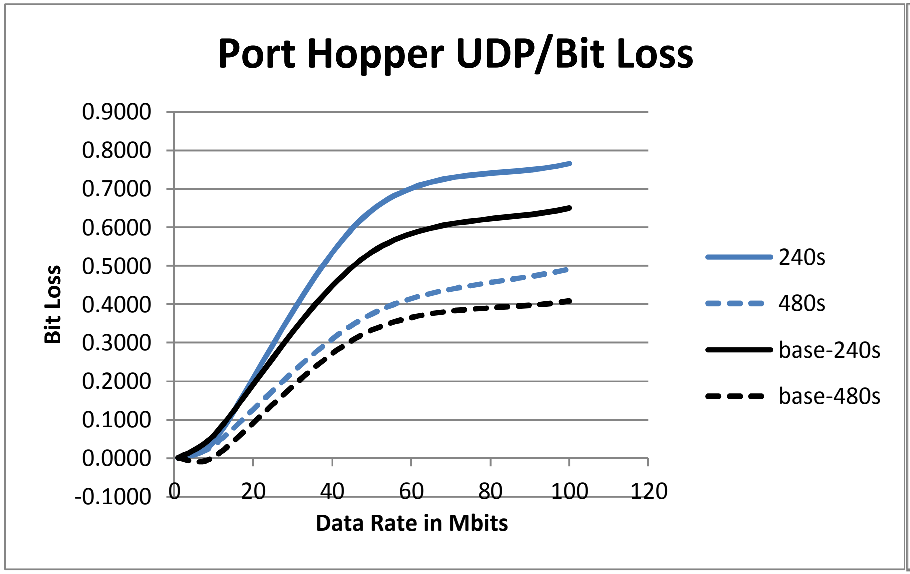
Test Duration (s)	Technique	BW Percentage
240	Port hopper	0.977628
480	Port hopper	0.981913
240	IP-proactive	0.998191
480	IP-proactive	0.998593
240	IP-reactive 10s	0.711685
	IP-reactive 30s	0.908663
	IP-reactive 60s	0.952302
	IP-reactive random	0.955762
480	IP-reactive 10s	0.724593
	IP-reactive 30s	0.910393
	IP-reactive 60s	0.967297
	IP-reactive random	0.953270

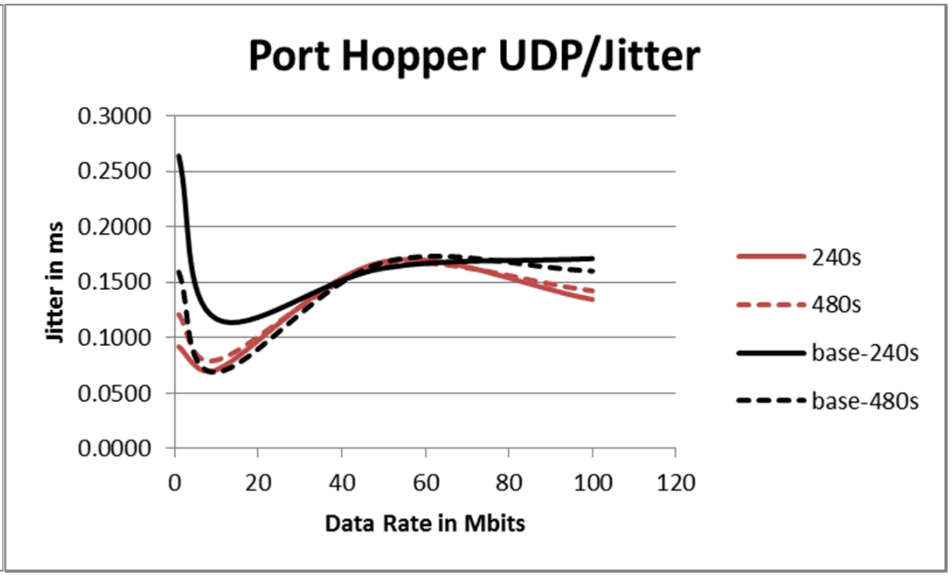
Experiment Results

No Randomization vs. Path Randomization

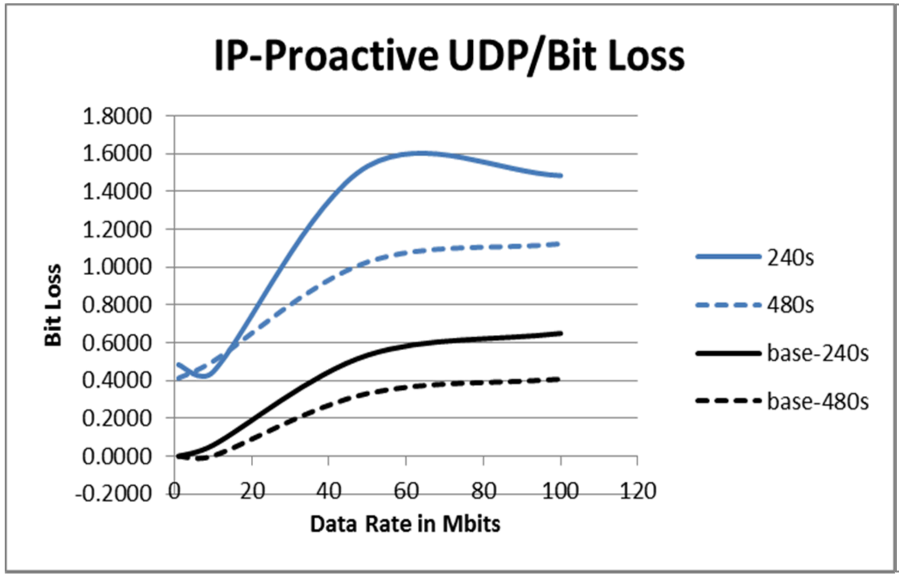
	RTT (ms)	10s Data Transfers (GB)	Bandwidth (Gbit/s)	1MB Transfer (ms)
Baseline	50.90798	22.25135	19.21674	69.11273
Random Path	62.64078	21.59769	20.59946	101.96778

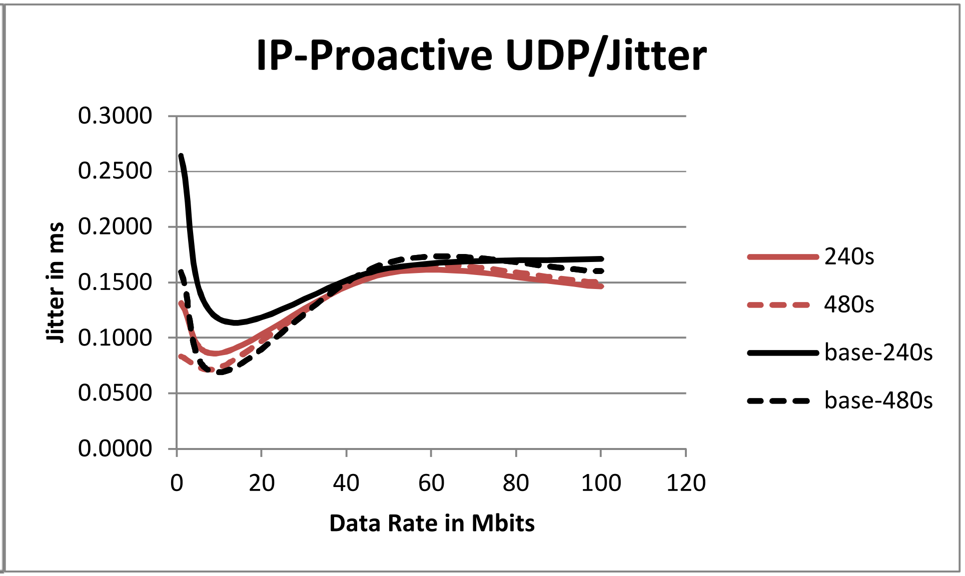
Experiment Results



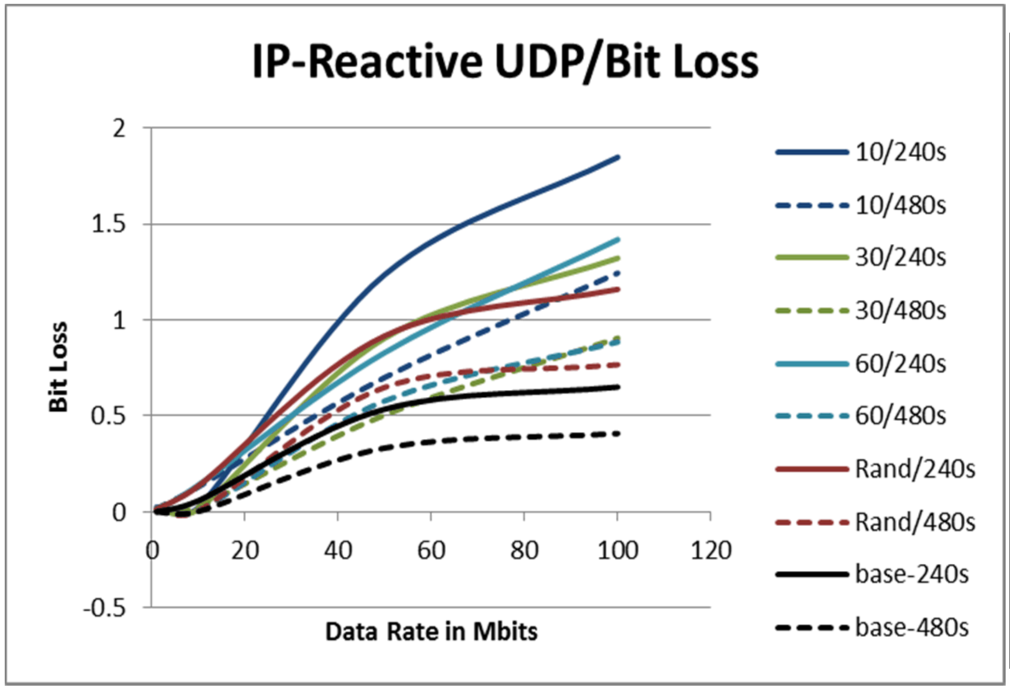


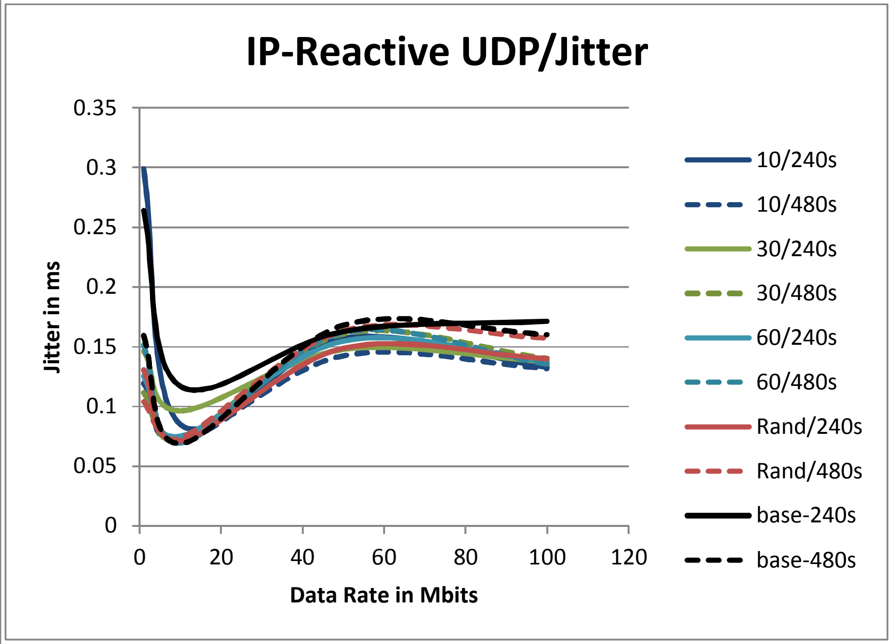
Experiment Results





Experiment Results





Conclusions

- Our experimental results show that the port randomization approach provides the least amount of performance impact, while successfully maintaining connectivity of a communication session.
- While port hopping works well to thwart application-based attacks, it still does not address protocols below TCP/UDP (e.g., ICMP). Addressing those types of attacks, as well as others mentioned, may be done with the IP Randomization techniques.

Conclusions

- The experimental results for the IP address randomization showed that while the two approaches successfully maintained connectivity between two hosts communicating with one another, impacts to performance were greater than the port hopper, most notably with the reactive IP address randomization approach.
- The advantage to the reactive IP address randomization approach is that resources and flow-rules are only used when communication is required; the reactive IP address randomization approach may suit well for low-bandwidth applications such as SCADA.

Conclusions

- For systems that require greater bandwidth coupled with delay-intolerance, the proactive IP address randomization approach should be considered.
- Path randomization should be used with care since additional hops through the overlay network may cause potentially unacceptable delays.
- In time critical applications, limiting the number of additional random hops in the overlay network should be considered.

Exceptional service in the national interest

Sandia National Laboratories

Questions/Comments

Techniques for the Dynamic Randomization of Network Attributes

William M.S. Stout

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration