

TITLE OF THESIS

How Long Does it Take to Offload Traffic from Firewall?

A Thesis

Presented to

the faculty of Engineering Technology

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

RajaRevanth Narisetty

December 2013

Acknowledgements

My sincere thanks to Dr.Deniz Gurkan for giving me this opportunity to work on Network Security and Network Management aspects of the emerging Software Defined Networking arena. My special thanks to Dr.Wajiha Shireen & Dr.Fatima Merchant for their valuable suggestions and inputs. I would like to thank Gregory Lebovitz, Gary Hemminger & Tony Chou of vARMOUR Networks Inc. for their insights and support during my research at University of Houston. I am thankful to all my friends for all their help and support which turned out to be an asset.

Contents

Chapter 1	12
Introduction.....	12
1. Network Firewall Placement.....	12
2. Virtual Firewalls	14
3. Distributed Firewalls.....	15
4. Software Defined Networking	15
5. OpenFlow Protocol	17
6. Software Defined Security [6]	20
Chapter 2	22
Background of the Study.....	22
1. Firewall Bottleneck.....	22
2. Firewall Throughput of Various Vendors in Market	22
3. Processes on Firewall.....	25
i. Deep Packet Inspection	25
ii. Intrusion Detection/Prevention System.....	25
4. Throughput Issues – Effects – Use Cases	26
i. Science DMZ [8].....	27
ii. Multi-site Use Case	28

Chapter 3	30
Overview	30
1. Application Identification with DPI.....	30
i. Study on Number of Packets to Identify Applications.....	31
2. Offload DPI of Classified Applications	31
i. Leverage SDN Paradigm for Offloading DPI	32
Chapter 4	33
Objective	33
Chapter 5	34
Framework	34
1. XEN Virtualization Platform	34
2. Network Elements.....	36
i. OpenFlow Switch.....	36
ii. vArmour SDSec Virtual Application	37
iii. vArmour Application Server.....	38
iv. Floodlight Controller.....	42
v. Tools.....	46
Chapter 6	47
OVS Based Experimental Setup	47
1. Virtualization Setup	47

i. Dom0 OVS & Networking.....	54
i. DomU's & Networking.....	54
ii. Data Plane & Control Plane (Data & Fabric Network).....	56
iii. Client & Server VMs.....	56
iv. Setup flows on OVS to pass C → S traffic via FW	56
v. Traffic Generation & Session on FW.....	57
vi. Session Offloading Flow Setup.....	58
2. Network Delays & Measurement Points.....	58
3. Measurements	60
4. Observations	60
5. Xen Virtualization Delays.....	61
Chapter 7	63
Setup with Open Flow Hardware Switch & Distributed VM's.....	63
1. Experimental Setup.....	63
Chapter 8	66
Network Delays	66
1. What is minimum length of a session on firewall to benefit from offloading?	66
i. Firewall to AppServer Network Delay.....	67
ii. AppServer Processing Delay.....	68
iii. AppServer to Floodlight Network Delay	68

iv. Floodlight Processing Time	69
v. Floodlight to OF Switch Network Delay	69
2. Determining Minimum Length of Session to Benefit from Offloading	70
Chapter 9	71
Measurements	71
1. Measurement Points on the Experimental Setup	71
2. Measurements	71
i. Firewall to AppServer network delay:	71
ii. AppServer Processing Time:.....	72
iii. Flow Setup Time:	73
iv. Floodlight Processing Time:	73
v. AppServer to Floodlight Network Delay:	74
Chapter 10	76
GENI Experimentation	76
1. GENI TOPOLOGY	76
2. Realization of Experimental Network Nodes on GENI.....	77
Chapter 11	78
Results	78
i. Firewall to AppServer Network Delay.....	78
ii. AppServer to Floodlight Network Delay:	79

iii. AppServer Processing Time:.....	81
iv. Floodlight to OVS Network Delay:	82
v. Floodlight Processing Time:	82
vi. How long does it take to identify SCP application on firewall?	84
Chapter 12	87
Conclusion	87
Chapter 13	92
Future Work.....	92
References	93
Appendix.....	97
1. GEC 16 POSTER.....	97
2. GEC 17 POSTER.....	98
3. GEC 18 POSTER.....	99
Bibliography	100
INDEX.....	107

List of Figures

Figure 1: Firewall Placement [1]	14
Figure 2: SDN Architecture [3]	16
Figure 3: Components of an OpenFlow switch [4].....	17
Figure 4: Sequence of OpenFlow messages between OpenFlow switch & controller [5]	19
Figure 5: Flow Mod Packet Structure	20
Figure 6: Science DMZ Architecture [8]	28
Figure 7: Multi-site Use Case	29
Figure 8: Available pCPUs on Xen Host.....	35
Figure 9: Memory share of VMs on Xen Host	36
Figure 10: AppServer Log	39
Figure 11: AppServer Controller Connection.....	39
Figure 12: AppServer connected Controller attached OpenFlow Switches	40
Figure 13: Default Flows on AppServer connected controller attached OpenFlow Switches	41
Figure 14: Offload Flow Rules configured on AppServer.....	41
Figure 15: Avior Controller Connection Interface.....	44
Figure 16: Avior – Controller connected switch flow rule summary	45
Figure 17: Avior - New flow rule setup Interface.....	45
Figure 18: Open vSwitch based Experimental Setup	47
Figure 19: Default Flow Rules on Open vSwitch.....	49
Figure 20: Pushing Static Flows	49
Figure 21: Data Plane.....	50

Figure 22: Firewall Configuration	50
Figure 23: Offload rule generation – Firewall – AppServer - User	51
Figure 24: Injection of Offload flow rules	51
Figure 25: Flow Mod Time Stamp.....	52
Figure 26: Offloaded rules	53
Figure 27: Data Plane - Offloaded Path	53
Figure 28: Xen Center – Virtual Network’s	54
Figure 29: Open vSwitch Controller Info	54
Figure 30: Open vSwitch Active Ports	55
Figure 31: Network Delay Plots	60
Figure 32: Experimental Setup with Hardware OpenFlow Switch	64
Figure 33: Offloading Events.....	66
Figure 34: Firewall to AppServer Network Delay.....	72
Figure 35: AppServer Processing Time	72
Figure 36: Sum of Floodlight Static Flow Processing Time and Floodlight to OVS network Delay	73
Figure 37: Floodlight Processing Delay	74
Figure 38: AppServer to Floodlight Network Delay	75
Figure 39: GENI Topology	76
Figure 40: Firewall to AppServer Network Delay.....	78
Figure 41: AppServer to Floodlight Network Delay {POST-OK} (b)	80
Figure 42: AppServer to Floodlight Network Delay TCP RTT (c)	81
Figure 43: AppServer Processing Time	81

Figure 44: Floodlight to OVS Network Delay.....	82
Figure 45: Floodlight Processing Time.....	83
Figure 46: Offload rules to OVS.....	84
Figure 47: SCP Application Identification Time on Firewall.....	85
Figure 48: Offload Delay Measurements (a)	85
Figure 49: Offload Delay Measurements (b)	86
Figure 50: Mean & 99th Percentile - Time to Offload - Virtualized Setup.....	88
Figure 51: Mean & 99th Percentile - Total Time to Offload - Physical PC Setup.....	90

List of Tables

Table 1: Firewall Throughput	25
Table 2: Virtual Machine Specifications	35
Table 3: OpenFlow Match/Action pairs on Floodlight REST API [16]	44
Table 4: VM/Port/Virtual Network Mapping	56
Table 5: Measurement Definitions.....	60
Table 6: Mean & 99th Percentile of Total Time to Offload on Virtualized Setup	88
Table 7: Mean & 99th Percentile of Total Time to Offload on Physical PC Setup.....	89

Chapter 1

Introduction

1. Network Firewall Placement

Firewalls comprise the principal network security elements of the enterprise/data center networks. They monitor and process all the ingress/egress traffic of the network against predefined policies, rules and existing set of virus, malware and worm signatures. Before proceeding further, it is essential to understand placement of the firewall in the network.

Placement of the firewall stands key in protecting the network from attacks, virus, worms, malware etc. There are many considerations and practices for firewall placement. Let us see the general practices of the firewall placement across the enterprise and data center networks.

i. Enterprise Networks

On the enterprise networks, firewalls stand as entry/exit points at the border router connecting to internet service provider ie at the corporate WAN edge [1]. Along with the actions -taken against pre-defined policies and rules, they perform operations like Deep Packet Inspection (DPI), intrusion detection/prevention.

Also, they may exist at multiple places in the network to monitor internal traffic. These include existing at the juncture of different internal enterprise networks, wired – wireless LAN perimeter etc. [1]

ii. Data Center Networks

As depicted in the Figure 1, when it comes to the data centers of an enterprise or the data centers of the service providers the following are the firewall placement practices:

- At the corporate WAN edge
- Between server farms
- Single or multiple instance per rack

Placement details of the firewall in the network (enterprise/data center) are depicted in the Figure 1. [1]

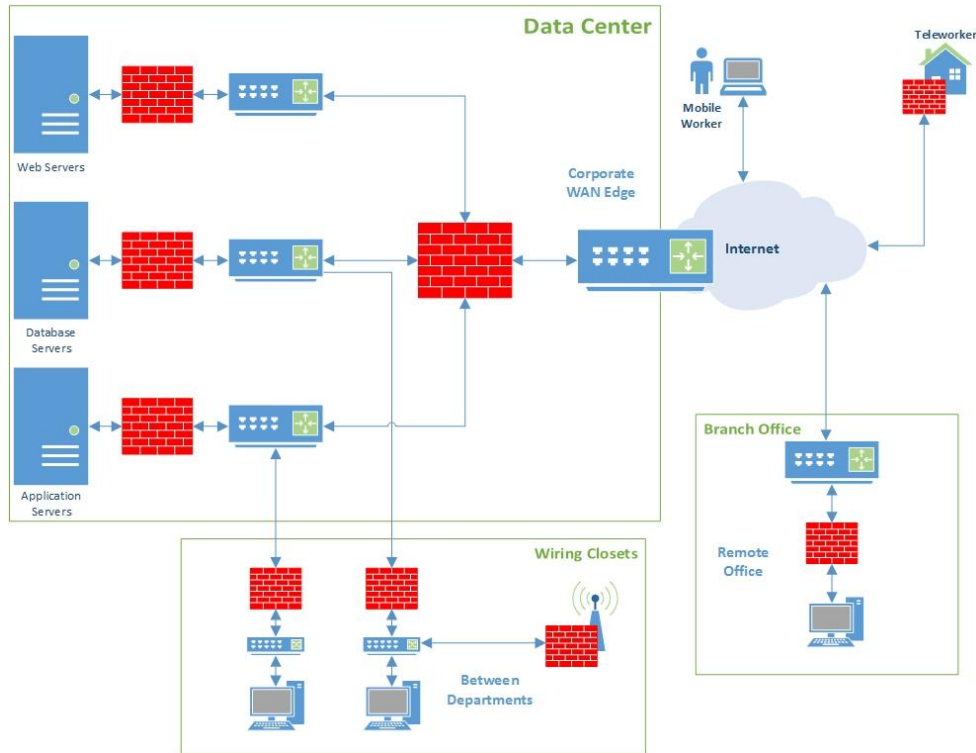


Figure 1: Firewall Placement [1]

2. Virtual Firewalls

Virtual Firewalls are the firewall instances running as virtual machines on a virtualized environment.

These monitor traffic of the tenants of the rack. The virtual firewall's can be implemented in one of the following ways: [2]

- Traditional software firewall on a guest virtual machine
- Virtual switch with additional security capabilities
- Custom built security appliance viewing network security requirements

Our discussion is in relevance to the third category of virtual firewalls which would be monitoring the ingress/egress traffic of the tenants of the data center rack.

3. Distributed Firewalls

Multiple firewall instances are deployed across the network which maintains state with a central firewall node. As the state is being maintained there exists track of the sessions of the virtual machine on all the instances of the firewall, even if tenant is migrated across. To a larger extent, virtual distributed firewalls mitigate issues related to firewall chokepoint, load balancing, dynamic allocation of security resources in need (new or unexpected workload).

4. Software Defined Networking

Unlike the legacy systems, software defined networking separates the decision making capabilities from a switching element/node. All the forwarding decision making capabilities are managed by piece of software called the OpenFlow Controller. One or many OpenFlow switches shall be attached to the OpenFlow Controller and this manages the flows pertaining to the switches. As the decision making is now centralized, there exists a complete network topology top down view at the decision making point which also brings in advantages during recovery.

The prime benefits of the SDN architecture are: [3]

- Cost Effectiveness, Dynamic and Adaptability
- Programmability, Agility etc
- Centralized control
- Programmatic configuration
- Open standards based and vendor neutral

The below diagram overview's the SDN abstraction: [3]

- Infrastructure layer – set of network resources like switches etc
- Control Layer – The above mentioned set of network resources are controlled in centralized manner through controller.
- Application Layer – Any custom or business specific requirements can be deployed.

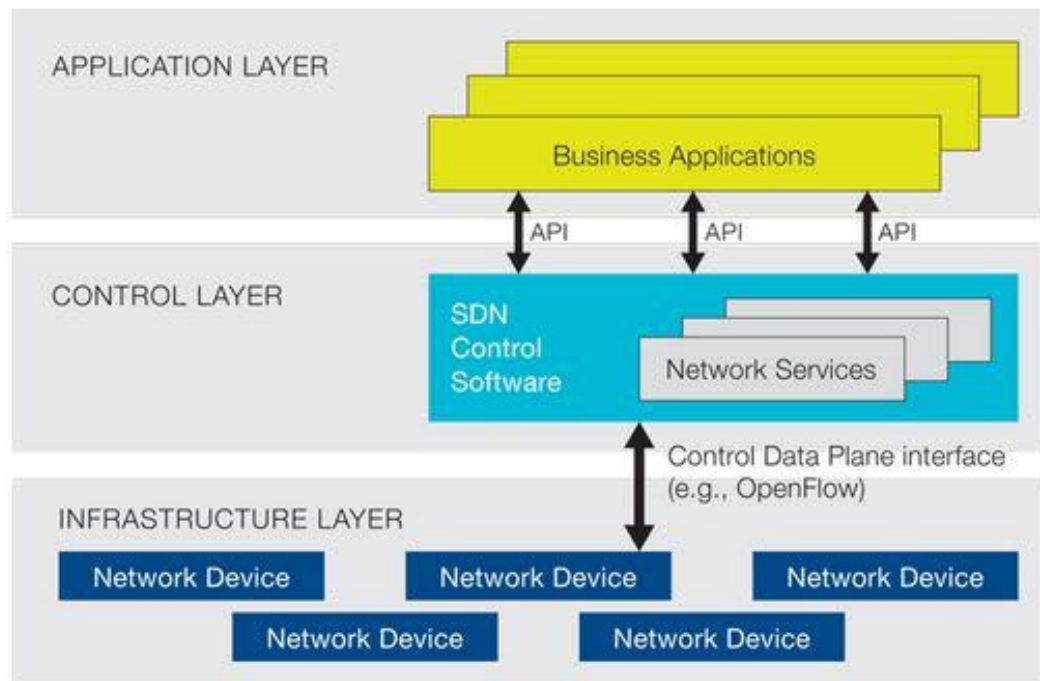


Figure 2: SDN Architecture [3]

5. OpenFlow Protocol

OpenFlow is the protocol on which the communication between OpenFlow switch and controller happens. OpenFlow enables us to set the path for data packets that ingress switch. These are updated in the flow tables as flow entries. Outbound decision for incoming traffic is taken after the flow table look up or by consulting controller in case of absence of relevant flow entry. [4]

Using the OpenFlow protocol, the controller can add, update, and delete flow entries in flow tables, both reactively (in response to packets) and proactively. Each flow table in the switch contains a set of flow entries; each flow entry consists of match fields, counters, and a set of ‘Actions’ (instructions to apply to matching packets). The above mentioned comprise the components of an OpenFlow switch and are depicted in the Figure 3. [4]

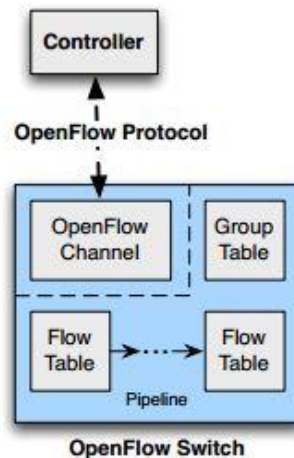


Figure 3: Components of an OpenFlow switch [4]

i. OpenFlow Protocol Messages

The OpenFlow protocol messages can be categorized as one of the following –

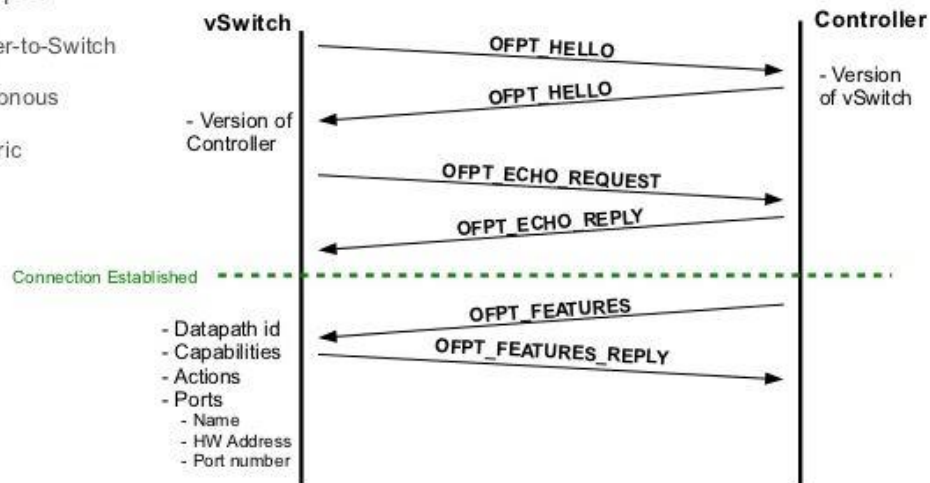
- Switch to controller messages
- Controller to switch messages

The charts explain about the types of OpenFlow message exchange sequences [5]

OpenFlow Initial Setup Protocol

Message Types

- Controller-to-Switch
- Asynchronous
- Symmetric



OpenFlow Protocol

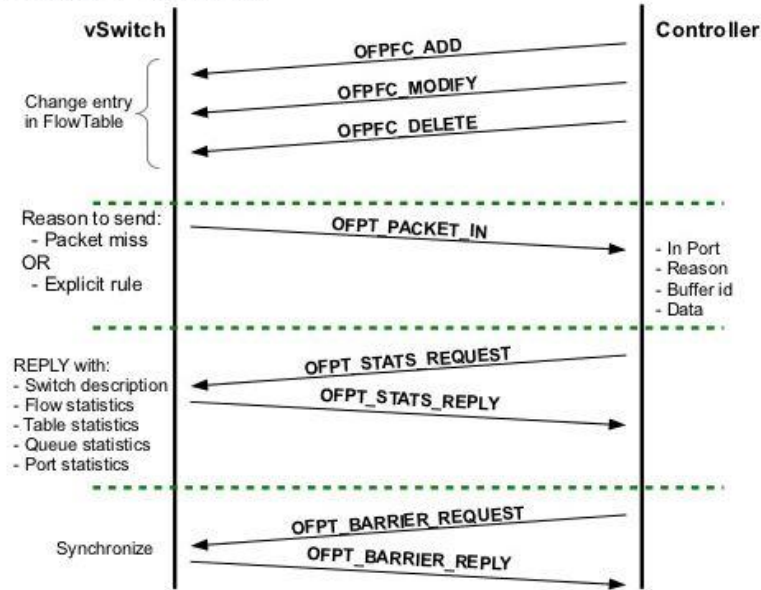


Figure 4: Sequence of OpenFlow messages between OpenFlow switch & controller [5]

The detailed packet structures and importance of all the messages are discussed in OpenFlow specifications by Open Networking Foundation.

In relevance to our experiment, we will discuss the ‘packet_in’ and ‘flow_mod’ messages where the earlier is from switch to the controller and later is the vice versa.

Here, we will quickly recall how OpenFlow based switches perform switching. [4]

- OpenFlow switch encapsulates as an OpenFlow message and forwards the first packet of a new flow to an OpenFlow controller attached.
- Controller communicates OpenFlow switch with the forwarding decision through ‘flow-mod’ or ‘packet-out’ message.
- Based on the controller’s decision the switch will forward the traffic.

The traffic/flow is identified based on match criteria specified in OpenFlow specification and the flow modification messages will be a set of match, action pairs.

The below is the sample packet trace of the OpenFlow flow-mod message on the wireshark, a network packet capture tool.

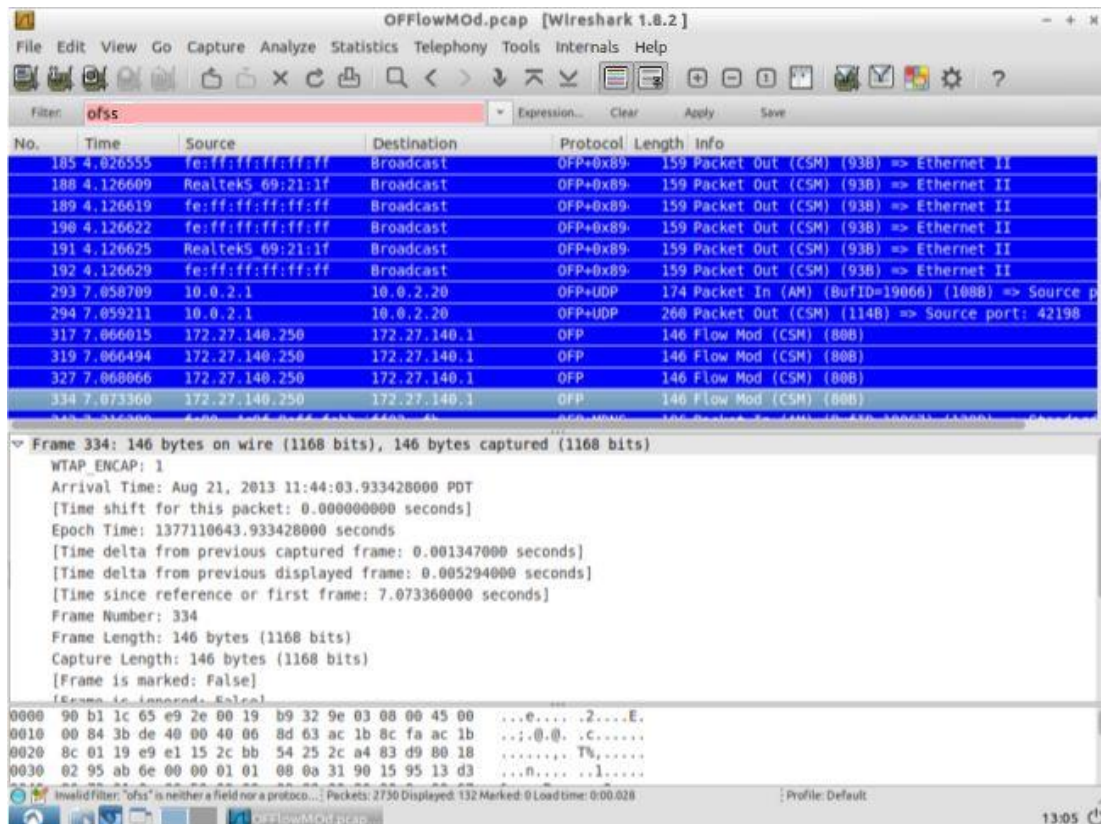


Figure 5: Flow Mod Packet Structure

6. Software Defined Security [6]

This section is an excerpt from the vARMOUR's Software Defined Security discussion and its benefits.

Similar to how SDN abstracts the network control plane from the forwarding plane, Software Defined Security (SDSec) separates the security control plane from the enforcement plane (data plane). This helps in building an efficient network security solution with the following features:

[6]

- Distributed Network Security Architecture – Multiple instances of the security appliances across the network maintaining state with the central node of the security system.
- Centralized Control – Abstraction of the instances, their current sessions and performance
- Easier Configuration Management – As all of the instances maintain state with the central control node and all of these control plane communication happens on the fabric network, all the instances are configurable from central control node.
- Load Distribution/Avoid Bottlenecks – The firewall load can be offloaded/distributed to other instances which are part of the distributed security solution on the fabric network.
- Effective Monitoring of VMs after/during migration – Consider a virtual machine, under the surveillance of an instance of the distributed security system, migrates to another host which is under surveillance of a different firewall instance. The earlier state/session information of the virtual machine are present in the new surveillance point because of the decoupled and distributed architecture of the security system.

Also the firewall instances (enforcement points) can be spun and placed across any required point of the network without losing any information and instant state information of all ongoing systems on the security system.

Chapter 2

Background of the Study

1. Firewall Bottleneck

Firewall's as part of deep packet inspection, inspect header and payload of all the ingress/egress traffic. Other firewall features like intrusion detection and prevention systems require huge processing capabilities. There is more probability of bottlenecks at the firewall irrespective of available higher bandwidths and network capabilities. In our research, we understood that throughput of available enterprise class firewall devices range between 700Mbps and 2Gbps. Also, upgrading the existing firewalls to achieve higher throughput ratios for the available bandwidth of the network to actual bandwidth of firewall involves huge cost.

2. Firewall Throughput of Various Vendors in Market

The below tabulation discusses the firewall throughput, deep packet inspection throughput, intrusion prevention system throughput etc. It is clearly evident that the DPI is the most bandwidth

consuming service amongst various firewall service's/features. Also, we all know how important it is to identify which application traffic it is to validate the traffic. DPI on every packet passing through the firewall would add much overhead on the load of the firewall. This would prove costly resulting in less overall firewall throughput despite the availability of higher capability ingress and egress links to the firewall.

This work adds a sense of intelligence to decide which traffic should undergo deep packet inspection which not only reduces load on the firewall but also helps with higher throughput to the safe traffic and also increases the ability of the firewall to scale to greater new connections at a point of time.

Vendor	Product	DPI Throug hput	Firewall Throug hput	IPS Throughput	Concurrent Sessions (BiD)	New Connecti ons/Sec
Dell SonicWall	NSA 4500	600Mbps	2.75gbps	1.4gbps	400000	10000
HP	F5000		40gbps		400k	180k
QOSMOS		10Gbps on LINUX				
Endace		10Gbps				

Cisco Systems	control engine 8000		30Gbps		10,00,000	max flow open rate 15 million flows per second
Palo Alto Networks	vm300		1gbps	600mbps	250000	8000
WatchGuard Technologies	XTM 515		2Gbps	1.6Gbps	40,000	24,000
	XTM 525		2.5Gbps	2Gbps	50,000	24,000
	XTM 535		3Gbps	2.4Gbps	1,00,000	28
	XTM 545		3.5Gbps	2.8Gbps	3,50,000	28,000
Juniper Networks	srx650		7gbps(large packets)	1gbps	512k	35,000
	idp8200		10gbps		5000000	
CyberRoam	Max of all models		1Gbps UDP & 0.7Gbps TCP	350	3,50,000	12000
Ipoque	Max of all models		75Gbps		6 million subscribers	
			10 million packets/second		240 million concurrent flows	

Table 1: Firewall Throughput

3. Processes on Firewall

Modern firewalls support majority of the features like NAT, Routing, Switching, VLANs and DHCP along with their own functionalities like deep packet inspection and intrusion detection/prevention. While the earlier features are additional features and later comprise the core functionalities of the firewall devices.

As part of the deep packet inspection, the firewalls process each packet's header and payload against a set of known signatures.

i. Deep Packet Inspection

Inspecting the header of the packet is Shallow Packet Inspection whereas inspecting the header and payload of the packet is Deep Packet Inspection. Traffic is processed and matched with an existing set of signatures.

ii. Intrusion Detection/Prevention System

The attempts to breach information security are prevalent. Intrusion Detection/Prevention Systems (IDS) are designed to detect and prevent any possible attack or anomalous activity in the network and the goal is to keep the network services intact.

The following are ideally the features of IDS [7]:

- Monitor/Analyze the user/system activity

- Auditing system configuration and vulnerabilities
- Assessing integrity of critical system and data files
- Statistical analysis of system activities
- Pattern match of known attacks

Typical IDS has the following components and their functionalities are outlined below [7]:

- Network Intrusion Detection System (NIDS)
 - Works in a promiscuous mode, and matches the traffic that is passed on the subnets to the library of known attacks.
 - Once the attack is identified, or abnormal behavior is sensed, the alert can be sent to the administrator.
- Network Node Intrusion Detection System (NNIDS)
 - NNIDS monitors traffic on the single host only and not for the entire subnet.
 - Example – Installing NNIDS on a VPN device, to examine the traffic once it was decrypted.
- Host Intrusion Detection System (HIDS)
 - Snapshot the host and matches for any modification/deletion or change in configuration of the host system

Here, in our case the IDS on firewall would be NNIDS. Also it works in conjunction with the Deep Packet Inspection.

4. Throughput Issues – Effects – Use Cases

In case of a firewall with above discussed features is deployed in the path of enterprise traffic, all the traffic will be inspected with DPI & IDS. This might cause unexpected delays, less bandwidth despite of the availability of higher networking capabilities and increased load on firewalls.

i. Science DMZ [8]

Science DMZ network architecture is tailored for higher performance applications distinct from general purpose network to benefit the educational/research groups. Stateful inspection and application-aware DPI on firewalls results in slower ingress and egress traffic rates, causing network bottlenecks when extremely high bandwidth sessions are flowing. Our demonstration of application-aware traffic steering addresses this issue by creating a fast path for science DMZ traffic to the OpenFlow switch as a flow definition for the remainder of the session once the application is identified. The measurements of delay elements for such a fast path redirection has not been reported within the context of a science DMZ to the scientific community. The session length will be compared with the total expected delay to illustrate the benefits and shortcomings of such a dynamic redirection mechanism. [8]

As depicted in the below architectural overview diagram, ScienceDMZ traffic will not pass through the enterprise border router or firewall. For separating the ScienceDMZ traffic from the enterprise traffic, a dedicated ScienceDMZ router/switch is employed. Thus the ScienceDMZ traffic is separated and efforts were taken to compromise security policies for fullest utilization of bandwidth and gain in network performance. [8]

Here as depicted in the Figure 6, instead of employing a dedicated ScienceDMZ router/switch we can utilize the SDN/OpenFlow paradigm for traffic separation and use the solution discussed in the forthcoming chapters for leap in the bandwidth for the ScienceDMZ traffic. After identification and separation we can offload ScienceDMZ traffic to the existing network

OpenFlow switch by maintaining a session on the firewall. For forwarding the traffic out of the enterprise that OpenFlow switch might need an uplink to the service provider or the destined networks.

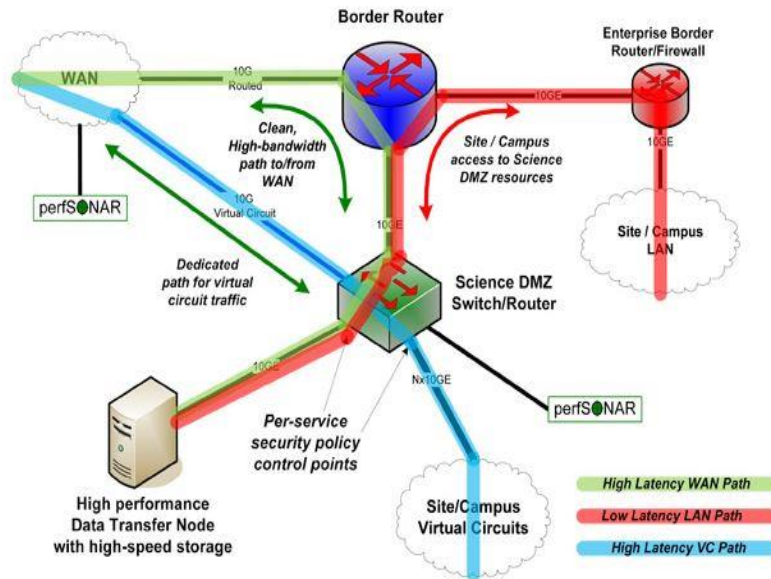


Figure 6: Science DMZ Architecture [8]

ii. Multi-site Use Case

In the following scenario as depicted in Figure 7, company 'X' has work sites at Houston and Atlanta. For the interconnectivity between the machines across these worksites, there exists an MPLS connectivity from the service provider with better Quality of Service. Network traffic egressing from work site at Houston/Atlanta is monitored by firewall at the corresponding network exit points. This remains true if the communication is across the work sites. It might not be essential to have the highest level of security enforcement for traffic between the work sites. As we discussed earlier in the '*Firewall Bottleneck*' section, firewall to perform deep packet inspection of all in/out traffic at line rate might not be possible and might effectively cause a bandwidth dip at the firewall. So, offloading deep packet inspection of trusted network

traffic (from worksite @ Houston to worksite @ Atlanta), trusted applications from firewall can benefit in avoiding bottlenecks at the firewall. This also benefits in providing line rate highest security policy enforcement to the un-trust (application) traffic.

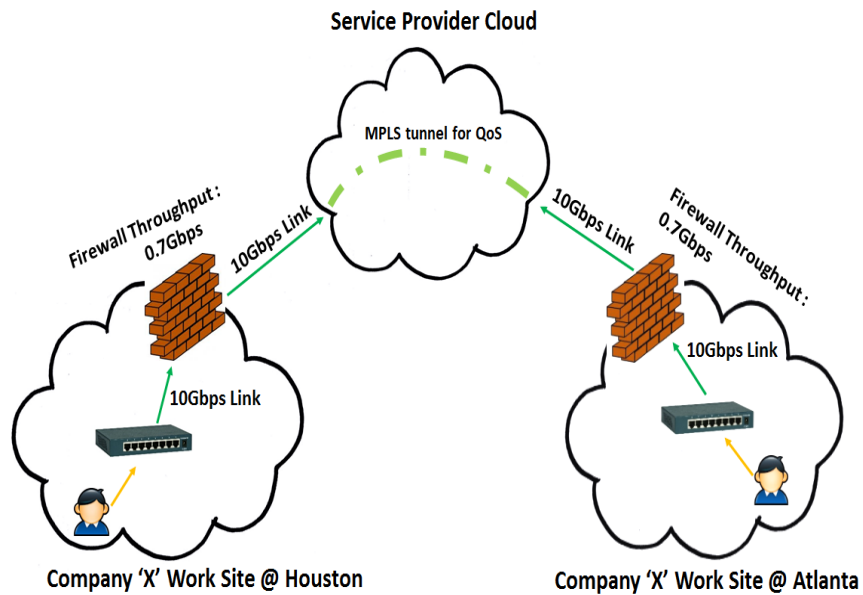


Figure 7: Multi-site Use Case

Chapter 3

Overview

1. Application Identification with DPI

Unlike protocol traffic, shallow packet inspection might not identify/classify application traffic.

Inspecting header and payload of a packet, DPI is required.

DPI is performed by one or more of the following methods [9]

- Well known port the application is using
- Pattern match with already existing database of identified signatures of applications, worms, virus etc.
- Statistics and/or behavioral study

It is expected that a firewall DPIs every ingress packet. This requires huge processing capabilities to maintain line rate with ongoing DPI. This results not only in a bottleneck but also huge difference in possible bandwidth and actual throughput of the device after DPI.

DPIBench is an industry accepted method for evaluating the performance of Deep Packet Inspection.[10] This approach benchmarks by classifying the tests into two classes – for

performance evaluation and validation whether the DPI choke point successfully stops the threats from entering the network.

i. Study on Number of Packets to Identify Applications

Known that DPI identifies application, it would be interesting to know approximate number of packets required to identify an application by DPI engine. It is understood that it varies with application and DPI vendor logic.

The study by ‘Performance of OpenDPI in Identifying Sampled Network Traffic’ by Jawad Khalife and Amjad Hajjar, quantifies approximate number of packets required to identify our daily use applications. Majority of the applications can be identified with less than 10 packets and protocols like iMESH and bit torrent takes more than 20 packets for identification [11].

2. Offload DPI of Classified Applications

We know that DPI is one of the prominent service running on most of the firewalls, which will positively identify the application. Also, this service consume lot of resources on the firewall as it requires huge processing capabilities which might also lead to bottlenecks as discussed in Section 1 of Chapter 2. To avoid such bottlenecks and improve the effective throughput of the firewalls, DPI of *trusted* applications can be offloaded. In the subsequent sections, we will discuss on leveraging SDN for such offload process. In the subsequent sections, we will

- Examine the sequence of events on firewall, OpenFlow switch and controller, which result in offloading load of trusted applications from the firewall.

- Conclude on the minimum length of the sessions on the firewall to benefit from such an offloading solution.

i. Leverage SDN Paradigm for Offloading DPI

Based on the fact that application will be positively identified by the DPI engine on firewall, application based configurations are done on the firewalls. We can use this application identification for redirecting the traffic (bypass traffic from being sent to firewall).

In this section, we will discuss how this application based traffic offload can be realized through SDN. With OpenFlow, the traffic forwarding can be done based on a set of match, action pairs as specified in OpenFlow specification.

We leverage firewall DPI to map all application traffic to OpenFlow flow rules. Traffic matching the application based rule set defined can be offloaded by translating them to a set of OpenFlow flow definitions. The same are pushed to the OpenFlow switch by the OpenFlow controller.

Chapter 4

Objective

‘What is the ideal length of session on the firewall to benefit from intelligent application steering based DPI offload solution?’

For the application traffic to benefit from this intelligent offload of deep packet inspection solution, determining the ideal length of the session on the firewall is the objective of this work. In other words, ‘How Long Does it Take to Offload Traffic from the Firewall?’ determines the ideal length of the session on the firewall for this intelligent offload solution.

Chapter 5

Framework

1. XEN Virtualization Platform

Xen is an x86 open source virtualization platform which allows us to create multiple virtual machines running different Operating Systems on a physical host [12]. For virtualization, Xen allows only driver domain or domain 0 to control all physical interfaces. All the guest virtual machines will communicate through the domain 0 to access physical NICs. Xen has driver set to access the physical NICs and a set of back-end interfaces to communicate with the guest domains. The back-end interfaces and the physical drivers are connected by a software bridge (linux bridge or Open vSwitch) inside the kernel of the domain 0. All communication from guest virtual machines will be through driver domain or domain 0 and vice versa. Virtual machines having pre-defined number of vCPUs and memory share physical resources like pCPUs and physical memory. Performance and network latency varies as per the sharing based on scheduling of Xen in background. [13]

In our experiment, we have the AppServer, Firewall, Client and Server as virtual machines on the host running Xen. The below tabulation mentions the resources allocated for the virtual machines.

The below resources are shared from the below available resources on the physical Xen host. Also the Figures 8 & 9 show the available processor and memory resources on our Xen host.

Virtual Machine	vCPU	Memory	Number of vNICs	Operating System
Firewall	4	8G	4(Min)	vAOS on Ubuntu
AppServer	2	2G	2	Ubuntu
Client	1 or 2	1G	1	Ubuntu
Server	1 or 2	1G	1	Ubuntu

Table 2: Virtual Machine Specifications

processor : 0	processor : 1
vendor_id : GenuineIntel	vendor_id : GenuineIntel
cpu family : 6	cpu family : 6
model : 58	model : 58
model name : Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz	model name : Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
stepping : 9	stepping : 9
cpu MHz : 3392.364	cpu MHz : 3392.364
cache size : 8192 KB	cache size : 8192 KB
physical id : 0	physical id : 1
siblings : 1	siblings : 1
core id : 0	core id : 0
cpu cores : 1	cpu cores : 1
apicid : 0	apicid : 1
initial apicid : 0	initial apicid : 1
fddiv_bug : no	fddiv_bug : no
hlt_bug : no	hlt_bug : no
f00f_bug : no	f00f_bug : no
coma_bug : no	coma_bug : no
fpu : yes	fpu : yes
fpu_exception : yes	fpu_exception : yes
cpuid level : 13	cpuid level : 13
wp : yes	wp : yes
flags : fpu de tsc mtrr pae mce cx8 apic sep mtrr	flags : fpu de tsc mtrr pae mce cx8 apic sep mtrr
aperfperf pni vmx est ssse3 hypervisor ida arat tpr_shad	aperfperf pni vmx est ssse3 hypervisor ida arat tpr_shad
bogomips : 6820.32	bogomips : 6820.32
clflush size : 64	clflush size : 64
cache alignment : 64	cache alignment : 64
address sizes : 36 bits physical, 48 bits virtual	address sizes : 36 bits physical, 48 bits virtual
power management:	power management:
processor : 2	processor : 3
vendor_id : GenuineIntel	vendor_id : GenuineIntel
cpu family : 6	cpu family : 6
model : 58	model : 58
model name : Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz	model name : Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
stepping : 9	stepping : 9
cpu MHz : 3392.364	cpu MHz : 3392.364
cache size : 8192 KB	cache size : 8192 KB
physical id : 2	physical id : 3
siblings : 1	siblings : 1
core id : 0	core id : 0
cpu cores : 1	cpu cores : 1
apicid : 2	apicid : 3
initial apicid : 2	initial apicid : 3
fddiv_bug : no	fddiv_bug : no
hlt_bug : no	hlt_bug : no
f00f_bug : no	f00f_bug : no
coma_bug : no	coma_bug : no
fpu : yes	fpu : yes
fpu_exception : yes	fpu_exception : yes
cpuid level : 13	cpuid level : 13
wp : yes	wp : yes
flags : fpu de tsc mtrr pae mce cx8 apic sep mtrr	flags : fpu de tsc mtrr pae mce cx8 apic sep mtrr
aperfperf pni vmx est ssse3 hypervisor ida arat tpr_shad	aperfperf pni vmx est ssse3 hypervisor ida arat tpr_shad
bogomips : 6820.32	bogomips : 6820.32
clflush size : 64	clflush size : 64

Figure 8: Available pCPUs on Xen Host

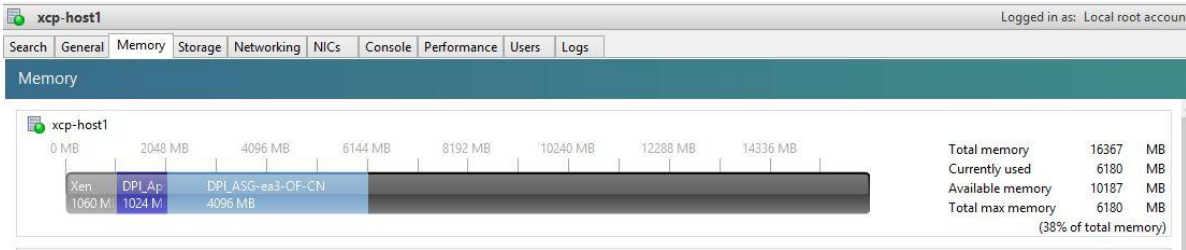


Figure 9: Memory share of VMs on Xen Host

Note – In Figure 9, memory allocation is shown for AppServer and Firewall virtual machines

2. Network Elements

In this section, network elements involved in the experimental topology are discussed – which include OpenFlow Switch, Firewall, AppServer, Floodlight Controller, Client and Server.

i. *OpenFlow Switch*

OpenFlow Switches use OpenFlow protocol to program the flow tables. These switches has three principal components – the flow table, secure channel to connect to the controller and OpenFlow protocol. [14] OpenFlow switches can be hardware or software switches. Let us discuss Open vSwitch & Pica8 in each category respectively.

a. *Open vSwitch*

Open vSwitch is defined as ‘a production quality, multilayer virtual switch designed to enable massive network automation through programmatic extension supporting standard

management interfaces and protocols (e.g. NetFlow, sFlow, SPAN, RSPAN, CLI, LACP, 802.1ag).’ [15]

In our experimental setup, the Xen Host1 has the Open vSwitch to switch traffic between the guest virtual machines. It maintains a flow table to look up and direct traffic accordingly. In case of no flow rule matching the incoming traffic, a packet-in is sent to the controller to which it is connected to and writes the packet-out/flow-mod into its flow table and switch traffic accordingly. Also, we can write the flow rules through static flow pushes to the controller or through ovs-ofctl tool set.

b. PICA8

Pica8 is a hybrid switch which operates in either OpenFlow or legacy modes or both simultaneously. The PICA8 OpenFlow switch is the hardware implementation of Open vSwitch. It is a switching platform with Debian linux on it to support custom applications development.

For our experiment, the Pica8 Pronto 3290 is used as the OpenFlow hardware switch in the Open vSwitch mode.

ii. vArmour SDSec Virtual Application

vArmour Virtual Firewall Appliance is a novel network security solution which provides:

- Capabilities to dynamically allocate and provision workloads
- Spread of virtualized security across the data center
- **Software Defined Security(SDSec)** solution – eliminates the provisioning, topology, performance and scaling bottlenecks that plague both traditional security solutions and

host-based virtual security solutions when applied in dynamic, agile, and often virtualized environments. [6]

- Inter-operability with traditional and software-defined networks.

For our experiment, we have raised an instance of vArmour virtual appliance as a virtual machine on the Xen Host. This security solution is built on Debian distribution of linux.

iii. vArmour Application Server

vArmour Application Server can be any machine (physical or virtual) which runs the vArmour AppServer java application. This acts as an interface between the security administrator and firewall for making the deep packet inspection offload decisions. Firewall communicates the information on the on-going sessions to the AppServer, which pushes static flow pushes to the controller based on the system administrator's application input on which application traffic to be offloaded.

For our experiment, we have raised an Ubuntu VM with the AppServer.jar running on it. The software provides us the below listed options. The Figure 10 shows the CLI of the AppServer and the debug logs that appear when offload rules are configured on the AppServer.

```
-a <file>      application map file name
-c <controller> controller url
-D <debug>     all | debug | info | error
-f <file>      flow file name
-l <file>      log file name
-n <network>   managed network
-p <password>  password of HTTP basic authentication
-r <file>      rule file name
-u <username>  username of HTTP basic authentication
```

This is invoked by

```
java -jar vArmourAppServer.jar -a applications.txt -D debug -c  
http://172.16.1.10:8080/ -t floodlight
```

```
2013-10-15 22:28:35,504 [pool-1-thread-157] DEBUG - Controller - RESP: 200  
227 * Client in-bound response  
227 < 200  
227 < Vary: Accept-Charset, Accept-Encoding, Accept-Language, Accept  
227 < Date: Wed, 16 Oct 2013 03:28:41 GMT  
227 < Content-Length: 27  
227 < Accept-Ranges: bytes  
227 < Connection: keep-alive  
227 < Content-Type: application/json; charset=UTF-8  
227 < Server: Restlet-Framework/2.1rc1  
227 <  
{"status" : "Entry pushed"}  
2013-10-15 22:28:35,505 [pool-1-thread-155] DEBUG - Controller - RESP: 200  
224 * Client in-bound response  
224 < 200  
224 < Vary: Accept-Charset, Accept-Encoding, Accept-Language, Accept  
224 < Date: Wed, 16 Oct 2013 03:28:41 GMT  
224 < Content-Length: 27  
224 < Accept-Ranges: bytes  
224 < Connection: keep-alive  
224 < Content-Type: application/json; charset=UTF-8  
224 < Server: Restlet-Framework/2.1rc1  
224 <  
{"status" : "Entry pushed"}  
2013-10-15 22:28:35,505 [pool-1-thread-158] DEBUG - Controller - RESP: 200  
226 * Client in-bound response  
226 < 200  
226 < Vary: Accept-Charset, Accept-Encoding, Accept-Language, Accept  
226 < Date: Wed, 16 Oct 2013 03:28:41 GMT  
226 < Content-Length: 27  
226 < Accept-Ranges: bytes  
226 < Connection: keep-alive  
226 < Content-Type: application/json; charset=UTF-8  
226 < Server: Restlet-Framework/2.1rc1  
226 <  
{"status" : "Entry pushed"}  
2013-10-15 22:28:35,507 [pool-1-thread-156] DEBUG - Controller - RESP: 200
```

Figure 10: AppServer Log

The screenshot shows the vArmour Networks Controller interface. At the top, there is a navigation bar with the vArmour Networks logo and three tabs: "Controller" (selected), "Networks", and "Rules". Below the navigation bar, the "Controller" section displays the following information:

- URL:** http://10.0.2.250:8080/
- API Version:** V1.0
- Application Count:** 748
- Rule Count:** 2

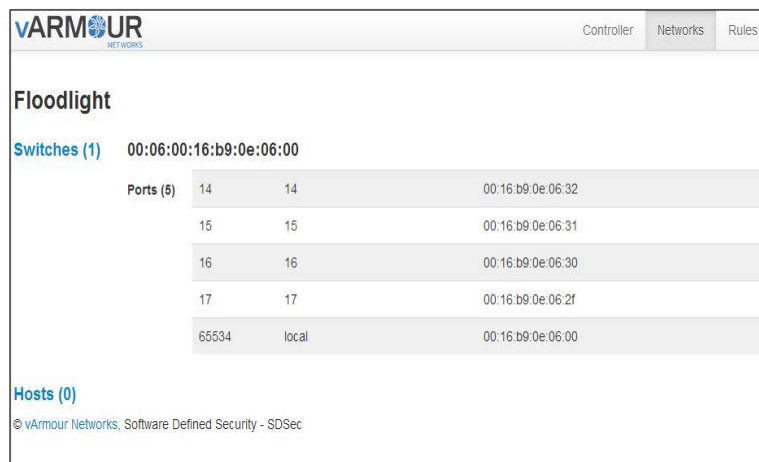
At the bottom of the interface, there is a copyright notice: "© vArmour Networks, Software Defined Security - SDSec".

Figure 11: AppServer Controller Connection

The Figure 11 displays the connectivity of the AppServer with the OpenFlow controller and this connection should remain active for pushing the new offload flow rules on to the OpenFlow controller which in turn pushes the same on to the attached OpenFlow switch.

Whereas the Figure12, displays information of the OpenFlow controller attached switches where we expect the new offload flow rules to reach. Also this interface fetches and displays information of the connected hosts and their attached ports on the OpenFlow switch.

Figure14 gives information about the interface to program the default rules which direct all the network traffic to the firewall. Figure 15 shows the interface to configure the safe application list and also the path in which we (network administrator) would like to redirect the required application traffic.



The screenshot shows the vARMOUR Networks Floodlight interface. At the top, there are tabs for 'Controller', 'Networks', and 'Rules'. The 'Networks' tab is selected. Below the tabs, the title 'Floodlight' is displayed. Under 'Floodlight', there is a section for 'Switches (1)' with a MAC address '00:06:00:16:b9:0e:06:00'. Below this, there is a table for 'Ports (5)' with 5 rows. The first four rows show ports 14, 15, 16, and 17, each with a corresponding MAC address. The fifth row shows port 65534 as 'local' with MAC address '00:16:b9:0e:06:00'. At the bottom, there is a section for 'Hosts (0)' and a copyright notice '© vArmour Networks, Software Defined Security - SDSec'.

Port	Port	MAC Address
14	14	00:16:b9:0e:06:32
15	15	00:16:b9:0e:06:31
16	16	00:16:b9:0e:06:30
17	17	00:16:b9:0e:06:2f
65534	local	00:16:b9:0e:06:00

Figure 12: AppServer connected Controller attached OpenFlow Switches

Controller Networks Rules

Default Flows

Network	Node	Port	→	Action	Node	Port	
Floodlight	00:06:00:16:b9:0e:06:00	14	→	REDIRECT	00:06:00:16:b9:0e:06:00	17	✕
Floodlight	00:06:00:16:b9:0e:06:00	15	→	REDIRECT	00:06:00:16:b9:0e:06:00	16	✕
Floodlight	00:06:00:16:b9:0e:06:00	16	→	REDIRECT	00:06:00:16:b9:0e:06:00	15	✕
Floodlight	00:06:00:16:b9:0e:06:00	17	→	REDIRECT	00:06:00:16:b9:0e:06:00	14	✕

Floodlight

00:06:00:16:b9:0e:06:00

14

→

PASS

00:06:00:16:b9:0e:06:00

14

✓

Reset Flows

Figure 13: Default Flows on AppServer connected controller attached OpenFlow Switches

Controller Networks Rules

Floodlight	00:06:00:16:b9:0e:06:00	15	→	REDIRECT	00:06:00:16:b9:0e:06:00	16	✕
Floodlight	00:06:00:16:b9:0e:06:00	16	→	REDIRECT	00:06:00:16:b9:0e:06:00	15	✕
Floodlight	00:06:00:16:b9:0e:06:00	17	→	REDIRECT	00:06:00:16:b9:0e:06:00	14	✕

Floodlight

00:06:00:16:b9:0e:06:00

14

→

PASS

00:06:00:16:b9:0e:06:00

14

✓

Reset Flows

Application Rules

Application	Network	Node	Port	→	Action	Node	Port	
http	Floodlight	00:06:00:16:b9:0e:06:00	16	→	REDIRECT	00:06:00:16:b9:0e:06:00	17	✕
http	Floodlight	00:06:00:16:b9:0e:06:00	17	→	REDIRECT	00:06:00:16:b9:0e:06:00	16	✕

0zz0

Floodlight

00:06:00:16:b9:0e:06:00

14

→

PASS

00:06:00:16:b9:0e:06:00

14

✓

Program Rules

© vArmour Networks, Software Defined Security - SDSec

Figure 14: Offload Flow Rules configured on AppServer

iv. Floodlight Controller

Floodlight Controller is an enterprise-class java-based OpenFlow controller. Let us discuss more about the Floodlight Controller in relevance to our experiment.

a. Static Flow Pusher API

Static Flow Pusher is a module, exposed via REST API, which allows us to program flows into one or more OpenFlow switches connected to it. [16]

b. Match Action Pairs

The OpenFlow standard defines the Match and Action pairs. There are various interfaces based on the OpenFlow Controller in use and they provide different interfaces to write flow entries on to the OpenFlow switches (one or more) dynamically. Here in our experiment we have used Floodlight Controller to manage OpenFlow switches in our topology. Floodlight Controller provides a REST API to push flows with these Match, Action combinations. Detailed documentation of the Static Flow Pusher REST API is available at <http://www.openflowhub.org/display/floodlightcontroller/Static+Flow+Pusher+API>. [16]

Out of available, we use only the below listed Match/Action pairs for our experiment.

Match field	Value	Notes
ingress-port	<number>	switch port on which the packet is received
		Hexadecimal or Decimal
src-mac	<mac address>	xx:xx:xx:xx:xx:xx

dst-mac	<mac address>	xx:xx:xx:xx:xx:xx
vlan-id	<number>	Hexadecimal or Decimal
vlan-priority	<number>	Hexadecimal or Decimal
ether-type	<number>	Hexadecimal or Decimal
tos-bits	<number>	Hexadecimal or Decimal
Protocol	<number>	Hexadecimal or Decimal
src-ip	<ip address>	xx.xx.xx.xx[/xx]
	[/mask]	
dst-ip	<ip address>	xx.xx.xx.xx[/xx]
	[/mask]	
src-port	<number>	Hexadecimal or Decimal
dst-port	<number>	Hexadecimal or Decimal
Key	Value	Notes
Output	<number>	no "drop" option
	all	(instead, specify no action to drop packets)
	controller	
	local	
	ingress-port	
	normal	
	flood	

Table 3: OpenFlow Match/Action pairs on Floodlight REST API [16]

c. Interfaces to Push Flows – CURL/AVIOR

CURL is a tool to transfer data from or to a server. cURL has a large of list of supported protocols like HTTP, FTP etc. The supported list protocols are on the cURL webpage. The command is designed to work without user interaction [17]. In our experiment, curl is widely used to push flows onto the OpenFlow network.

Whereas AVIOR gives us a user interface to push flows manually [18]. Figure 15 depicts the interface to connect to the Floodlight Controller. Figure16 displays the screen where we can fetch all the information about the Floodlight Controller attached OpenFlow switches, active ports on the same, interface statistics and also the active flows on the switch. Whereas the Figure 17 shows the interface to configure and push new flow rules on to the OpenFlow switch of your choice which is attached to the Floodlight Controller.



Figure 15: Avior Controller Connection Interface

Overview for switch : 00:06:00:16:b9:0e:06:00

Avior v1.2

Manufacturer: HP-Labs

Hardware : HP - Switch E3500yl-48G - J8693A

Software : 2.02w

Serial Number : SG704TH006

Ports

#	Link Status	Tx Bytes	Rx Bytes	Tx Pkts	Rx Pkts	Dropped	Errors
14	UP - 1 Gbps FDX	2130 MB	2451 MB	1908 K	2012 K	0	-3
15	UP - 1 Gbps FDX	2459 MB	2115 MB	2083 K	1839 K	0	-3
16	UP - 1 Gbps FDX	4775 MB	3212 MB	3660 K	2753 K	0	-3
17	UP - 1 Gbps FDX	3214 MB	4781 MB	2777 K	3633 K	0	-3
-2	UP	-1	-1	-1	-1	-2	-5

Flows

#	Priority	Match	Action	Packets	Bytes	Age	Timeout
1	0	port:15	output:16	352	0	55342	Static
2	0	port:14	output:17	4440	0	55342	Static
3	-32768	ether-type:0x0800, port:16, dst-ip:11.11.1.5, src-ip:11.11.1.6, dst-port:80, src-p...	output:17	4295	0	54932	Static
4	-32768	ether-type:0x0800, port:16, dst-ip:11.11.1.6, src-ip:11.11.1.5, dst-port:17990, ...	output:17	0	0	54932	Static
5	-32768	ether-type:0x0800, port:16, dst-ip:11.11.1.5, src-ip:11.11.1.6, dst-port:80, src-p...	output:17	35	0	54807	Static
6	-32768	ether-type:0x0800, port:16, dst-ip:11.11.1.6, src-ip:11.11.1.5, dst-port:17988, ...	output:17	0	0	54807	Static
7	-32768	ether-type:0x0800, port:16, dst-ip:11.11.1.6, src-ip:11.11.1.5, dst-port:17988, ...	output:17	0	0	54796	Static
8	-32768	ether-type:0x0800, port:16, dst-ip:11.11.1.5, src-ip:11.11.1.6, dst-port:80, src-p...	output:17	4215	0	54796	Static
9	0	port:16	output:15	145	0	55342	Static
10	-32768	ether-type:0x0800, port:17, dst-ip:11.11.1.5, src-ip:11.11.1.6, dst-port:80, src-p...	output:16	0	0	54932	Static
11	-32768	ether-type:0x0800, port:17, dst-ip:11.11.1.6, src-ip:11.11.1.5, dst-port:17990, ...	output:16	0	0	54932	Static
12	-32768	ether-type:0x0800, port:17, dst-ip:11.11.1.6, src-ip:11.11.1.5, dst-port:17988, ...	output:16	102	0	54807	Static

Figure 16: Avior – Controller connected switch flow rule summary

Overview for switch : 00:06:00:16:b9:0e:06:00

Floodlight Static Flow Manager

Avior v1.2

File Help

Switches

00:06:00:16:b9:0e:06:00

Refresh New Flow Push Clear Delete Flow Delete All Flows

Parameter Value

Name

Actions

Match

Priority

Flows

FLOW-906

FLOW-907

FLOW-908

FLOW-909

FLOW-910

FLOW-912

FLOW-911

FLOW-914

FLOW-905

FLOW-913

FLOW-916

flow-mod-1

FLOW-915

flow-mod-2

flow-mod-3

flow-mod-4

Figure 17: Avior - New flow rule setup Interface

v. *Tools*

a. *OpenFlow Dissector Wireshark Module*

It is not possible to dissect the OpenFlow packets with the regular wireshark application as the libraries for dissecting OpenFlow protocol messages are not built with-in. We need to compile by downloading the source available at the below link

http://archive.openflow.org/wk/index.php/OpenFlow_Wireshark_Dissector

In this experiment the measurements are based on the OpenFlow wireshark dissector. Tshark is the CLI based interface for wireshark.

b. *Tcpdump*

Tcpdump is a packet capture application generally built within the kernel for all Linux distributions [19].

Chapter 6

OVS Based Experimental Setup

1. Virtualization Setup

Our experimental setup is built on the XEN virtualization platform. Client, Server, Firewall, AppServer virtual machines (VMs) are hosted on Xen. Floodlight Controller is hosted separately as a standalone server on the network.

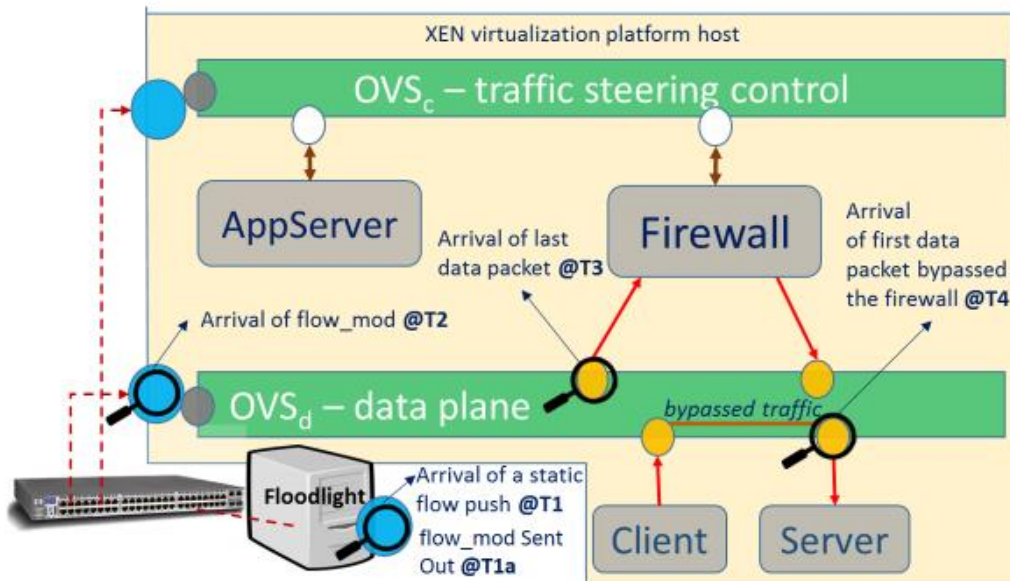


Figure 18: Open vSwitch based Experimental Setup

The various network servers/components depicted in the above diagram are explained hereunder.

In this context, here are the main components of our measurement setup:

- Client is Ubuntu VM communicating with a server VM using SCP (secure copy protocol) and HTTP applications
- Server is a web server on an Ubuntu VM
- Firewall is a distributed virtual firewall appliance fabric from vARMOUR Networks, Inc. supporting OpenFlow protocol. Note that vArmour appliance has a firewall rule setup that enables firewall bypass through a “permit steering” configuration for preferred DPI traffic.
- Controller is a server on the network that is running Floodlight
- AppServer is a basic static flow push application for the preferred DPI traffic, hosted on a VM on the network. Note that this application keeps track of all ongoing sessions on the firewall and passes the required session information to Floodlight with the translation of that particular session into an OpenFlow flow definition.

The experiment has the following expected events & each event is diagrammatically depicted:

- Client packets start being transmitted with DPI in the firewall. Note that OVS that is forwarding traffic to/from firewall has been pre-configured with the particular client/server flows so that firewall is the choke point for any sessions of the client/server.

```
[root@xcp-host1 ~]# ovs-ofctl dump-flows xenbr1
NXST_FLOW reply (xid=0x4):
 cookie=0xa0000000000000, duration=633527.84s, table=0, n_packets=23202, n_bytes=1404743, priority=0,in_port=60 actions=output:100
 cookie=0xa0000000000000, duration=633527.818s, table=0, n_packets=23207, n_bytes=1405130, priority=0,in_port=61 actions=output:103
 cookie=0xa0000000000000, duration=633527.851s, table=0, n_packets=499, n_bytes=46059, priority=0,in_port=100 actions=output:60
 cookie=0xa0000000000000, duration=633527.829s, table=0, n_packets=491, n_bytes=45357, priority=0,in_port=103 actions=output:61
```


Figure 19: Default Flow Rules on Open vSwitch

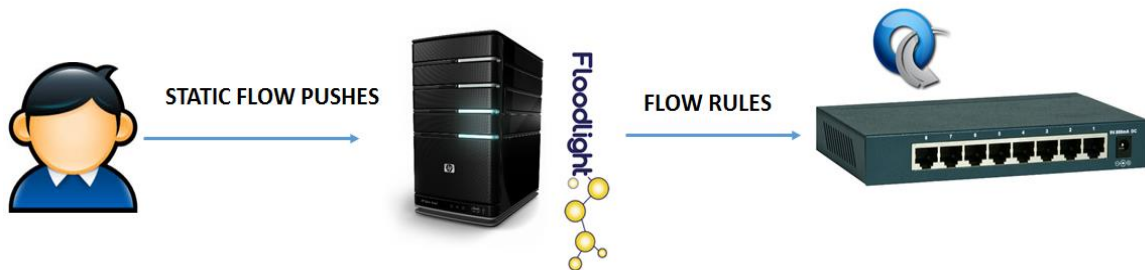


Figure 20: Pushing Static Flows

- After a few packet transmissions through the DPI module, an emulated classified application is identified by the firewall DPI function. We assumed SCP and HTTP in this experiment. However, there are many other protocols and applications that can be identified and translated into an OpenFlow flow definition.

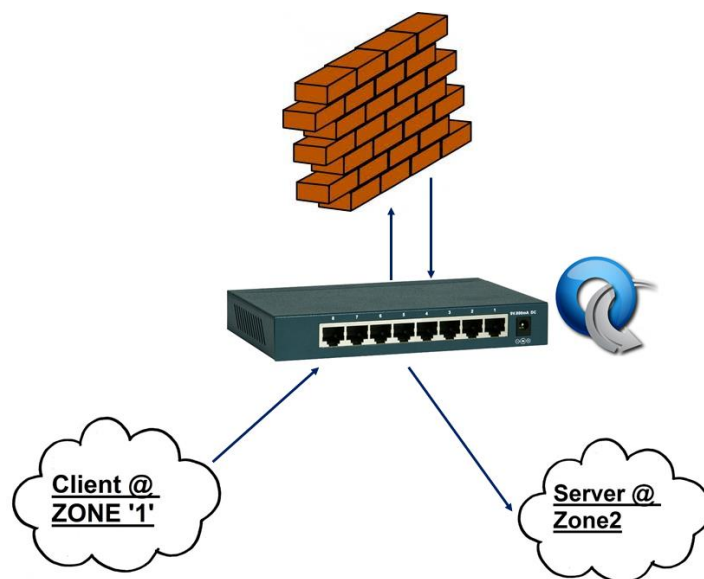


Figure 21: Data Plane

- Firewall is configured to invoke the ‘permit steering’ rule for all applications that are identified as being in a session. The rule configuration at the firewall is:

```
set policy z1-z2 zone z1 z2 rule PERMIT-ALL action permit steering app-server
```

Where z1 and z2 represent zones from which the classified traffic is coming in and sent out respectively.

```
varmour@vvarmour> show running-config
!## last commit: 2013-09-22 20:03:49 UTC by Current User
set system user varmour role admin
set system user varmour password $6$5ZzI2HSI$SnDgDUJgweYnA7jwcdWFwFpDeW3KE9tOpFIJ89i.RZeBFgjxULWCNuBCepjAug3DrICNfYki/PM4kfDzPiaEH/
set zone z1 type L2
set zone z2 type L2
set zone z1 interface fe-1/0/4.0
set zone z2 interface fe-1/0/5.0
set policy z1-z2 zone z1 z2 rule PERMIT-ALL match source-address Any
set policy z1-z2 zone z1 z2 rule PERMIT-ALL match destination-address Any
set policy z1-z2 zone z1 z2 rule PERMIT-ALL match service Any
set policy z1-z2 zone z1 z2 rule PERMIT-ALL match app-group Varmour::Any
set policy z1-z2 zone z1 z2 rule PERMIT-ALL action permit steering app-server
set profile intrusion-detection ID icmp-flood enable
set profile stateful-inspection SD strict-top-check enable
set profile app-steering app-server server-ip-address 10.0.2.20
set profile app-steering app-server server-port 30273
set profile app-steering app-server server-protocol udp
set profile app-steering app-server resend-interval 3
set profile app-steering app-server session-timeout 30
```

Figure 22: Firewall Configuration

- AppServer gets updates from the firewall about the ongoing sessions on the firewall.
- On the AppServer UI, the required action pertaining to the offload of the identified/classified application is to be configured.

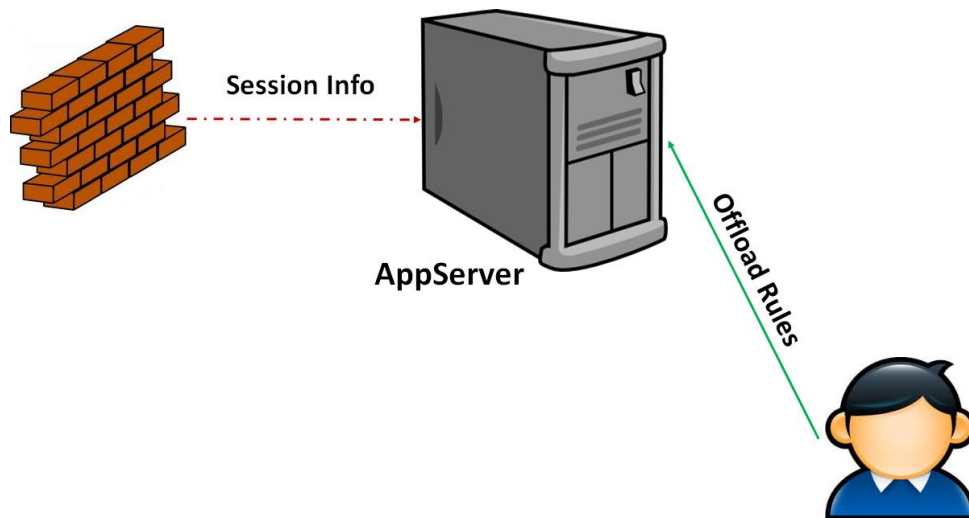


Figure 23: Offload rule generation – Firewall – AppServer - User

- AppServer translates the received session information as update from the firewall and the above mentioned action configured into a static flow push message for the controller.

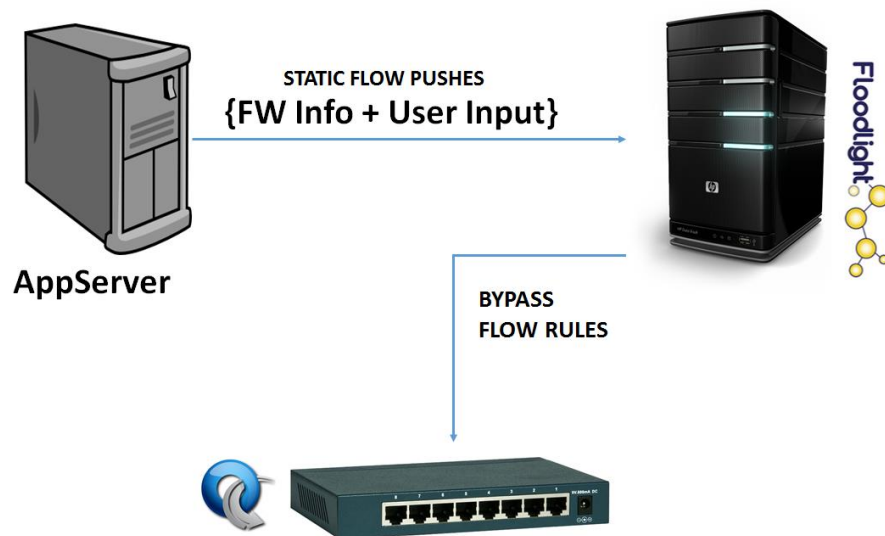


Figure 24: Injection of Offload flow rules

- A static flow push OpenFlow message is sent from Floodlight to the corresponding OpenvSwitch (OVS). Note that this flow push would initiate the bypass of the firewall for the particular application traffic.

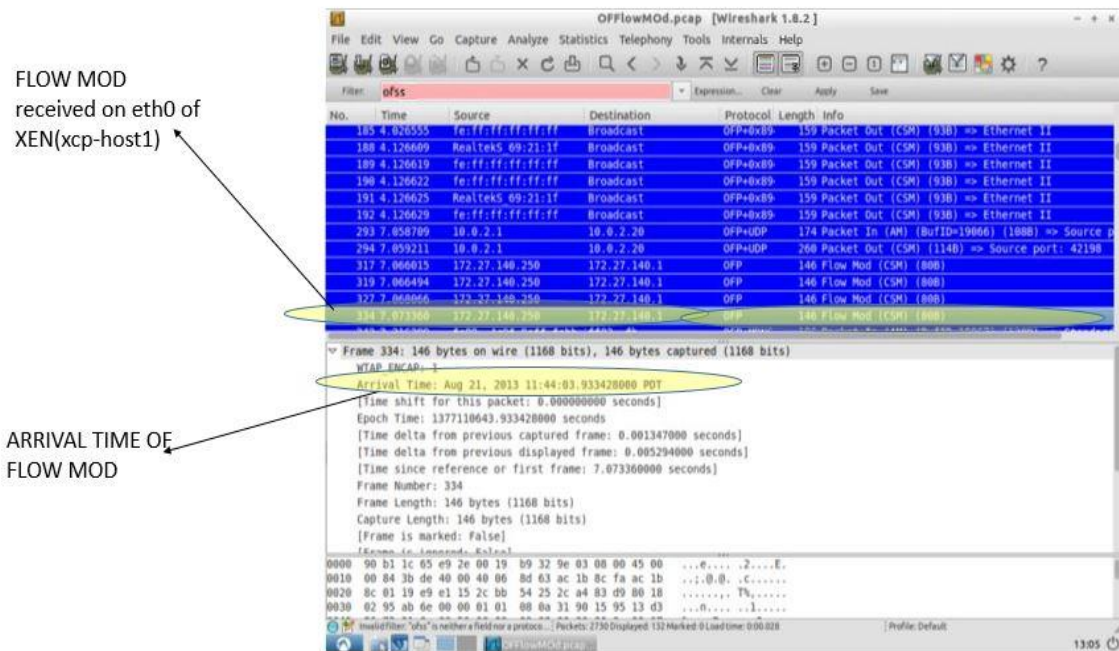


Figure 25: Flow Mod Time Stamp

- The identified/classified traffic goes through the OVS and bypasses the firewall with the new flow rule and goes directly to the destination server.

```
cookie=0xa0000000000000, duration=633514.396s, table=0, n_packets=3560, n_bytes=258736, tcp,in_port=103,nw_src=11.11.1.2,nw_dst=11.11.1.1,tp_src=22,tp_dst=36213 actions=output:100  
cookie=0xa0000000000000, duration=633514.396s, table=0, n_packets=6403, n_bytes=53006878, tcp,in_port=100,nw_src=11.11.1.1,nw_dst=11.11.1.2,tp_src=36213,tp_dst=22 actions=output:103
```

Figure 26: Offloaded rules

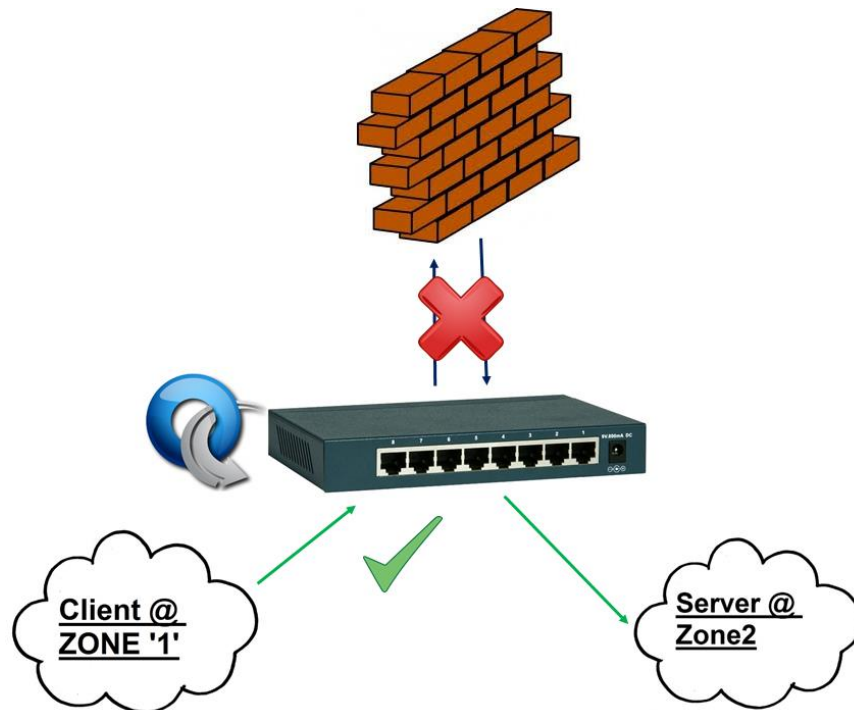


Figure 27: Data Plane - Offloaded Path

i. Dom0 OVS & Networking

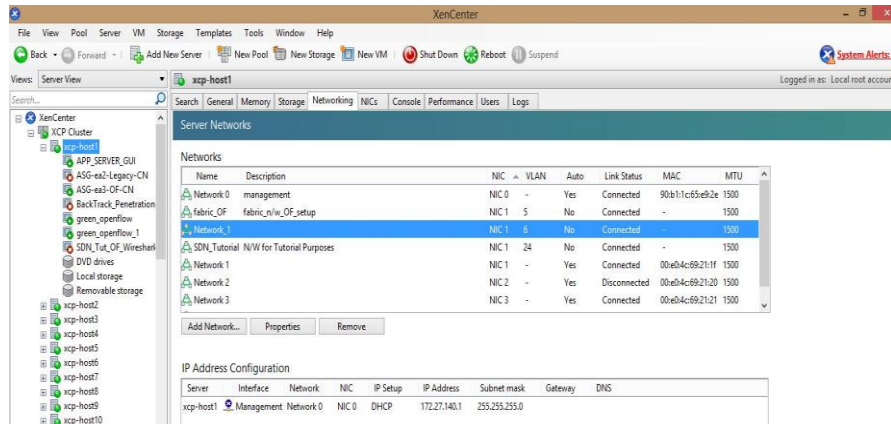


Figure 28: Xen Center – Virtual Network's

The Open vSwitch instance is attached to the Floodlight Controller in the network.

```
[root@xcp-host1 ~]# ovs-vsctl get-controller xenbr1
tcp:172.27.140.186:6633
```

Figure 29: Open vSwitch Controller Info

i. DomU's & Networking

The attachment points of the guest virtual machines to the OVS instance on the Dom 0 are as below & the below tabulation explains the attachment points of various DomU's involved in our experiment:

```

[root@xcp-host1 ~]# ovs-dpctl show xenbr1
system@xenbr1:
  lookups: hit:7550423396 missed:580895572 lost:0
  flows: 0
  port 0: xenbr1 (internal)
  port 1: eth1
  port 2: xapi0 (internal)
  port 3: xapi1 (internal)
  port 57: vif11.0
  port 58: vif11.2
  port 59: vif11.3
  port 60: tap11.2
  port 61: tap11.3
  port 62: tap11.0
  port 100: vif29.0
  port 103: vif30.0
  port 106: vif31.0
  port 114: xapi2 (internal)
[root@xcp-host1 ~]#

```

Figure 30: Open vSwitch Active Ports

Attachment Point: xenbr1					
Virtual Machine	Port#OVS	Virtual Network	VLAN	Connectivity VMs	Remarks
Client	vif29.0: 100	Network_1	6	Server, Firewall in_port	Data Network
Server	vif30.0: 103	Network_1	6	Client, Firewall out_port	Data Network
AppServer	vif31.0: 106	fabric_OF	5	Firewall	Fabric/Control/ Steering Network
Firewall	In: tap11.2: 60	Network_1	6	Server,Client	Data Network
	Out: tap11.3: 61	Network_1	6	Server,Client	Data Network

	Fabric Int:	fabric_OF	5	AppServer	Fabric/Control/ Steering Network
--	-------------	-----------	---	-----------	--

Table 4: VM/Port/Virtual Network Mapping

ii. Data Plane & Control Plane (Data & Fabric Network)

Data Plane is where the Client and Server communication happens. In our experimental setup, Network_1 comprises of the Data Plane and is called as Data Network.

Control Plane in our experimental setup will comprise of the ‘fabric_OF’ network. It connects the fabric interface of the Firewall and the AppServer. Firewall updates AppServer of the ongoing sessions through the fabric network.

iii. Client & Server VMs

The Client and Server machines are the Ubuntu virtual machines raised on the XEN host and are connected to the ‘Network_1’ i.e., the data network. These are isolated from the Control plane i.e., the ‘fabric_OF’ network. These VMs have the Apache web server installed in them and also are capable of generating the SCP, VOIP traffic etc.

iv. Setup flows on OVS to pass C → S traffic via FW

Initially, traffic between Client and Server VMs not necessarily pass through the Firewall, as both the VMs are bridged under ‘OVS-d’ in Figure 30. The controller assigned to the bridge or the default flow with action ‘NORMAL’ will establish reachability. i.e., the incoming Client traffic on port number 100 will be forwarded through ‘103’ port on OVS-d and vice versa.

We need the traffic between Client and Server (and vice versa) to be monitored. For the same we have pushed flows on to the OVS-d to pass the traffic through the firewall. i.e., the incoming Client traffic on port 100 is forwarded through the firewall connected port 60 and the egress traffic from the firewall connected port 61 is forwarded through the server connected port 103 and vice versa. Here, the firewall operates in layer2 mode.

The above mentioned flows can be made available to switch in two ways:

Sample static flow pushes through the controller:

```
curl -d '{"switch": "00:06:00:16:b9:0e:06:00", "name": "flow-mod-1", "priority": "0",  
"ingress-port": "100", "active": "true", "actions": "output=60" }'
```

```
http://172.27.140.250:8080/wm/staticflowentrypusher/json
```

```
curl -d '{"switch": "00:06:00:16:b9:0e:06:00", "name": "flow-mod-4", "priority": "0",  
"ingress-port": "61", "active": "true", "actions": "output=103" }'
```

```
http://172.27.140.250:8080/wm/staticflowentrypusher/json
```

Sample flow rules on the Open vSwitch through the ovs-ofctl tool kit:

```
ovs-ofctl add-flow xenbr1 in_port=100,actions=output:60
```

```
ovs-ofctl add-flow xenbr1 in_port=60,actions=output:100
```

v. Traffic Generation & Session on FW

We generated http & scp traffic between the Client and Server passing through the firewall.

SCP: Secure Copy protocol uses Secure Shell for file transfer between hosts on a network. It runs on TCP port 22. A big file transfer is monitored between the Client and Server. The measurements were made in iterations of this process.

HTTP: Web traffic is generated by installing a web server in the Server VM and the traffic is monitored through firewall. The measurements were made in iterations of this process.

vi. Session Offloading Flow Setup

The sessions between Client and Server are being monitored at the firewall. As mentioned in the earlier sections, every packet of the session will be deep packet inspected. Here, in case of higher bandwidth requirements for known/trusted traffic sessions, we can employ the ‘Intelligent DPI Offloading through Application Steering’. We can offload the session on to the Open vSwitch to enable direct communication between Client and Server (bypassing firewall).

In the AppServer UI, the trusted applications are to be selected and offload rules can be pushed. The firewall communicates about the on-going sessions. AppServer uses this information in conjunction with user selected applications, during the Offload rules flow push.

2. Network Delays & Measurement Points

It is important to find the below delay’s, which contribute to ‘*time to offload*’ a deep packet inspection of an application. The below listed delays and the measurement points are defined in the table 5. The corresponding T1,T1a, T2, T3 & T4 refer to the Figure18. Please note that some of these definitions are limited to the experiment based on the virtualized setup only.

- Controller setup time
- Static Flow Processing Time
- OVS Setup Time
- Controller to OVS Network Delay

Measurement Points	
T1 @Floodlight – Static Flow Push Arrival Time	T1a @Floodlight - Flow Mod Sent Out Time
T2 @OVS – Arrival of Flow Mod	T3 @OVSD - Arrival of last data packet on Firewall
T4 @OVSD - Arrival of first offloaded packet onto the Server connected port of OVSD	
Delay	Measurement
Static Flow Processing Time	Difference between the arrival time of Static Flow (HTTP POST) from AppServer and Flow Mod Sent Out time i.e., (T1a-T1)
Floodlight to OVS Network Delay	Difference between the Arrival Time of FlowMod on OVS and Flow Mod Sent out Time on Controller (T2 - T1a)
Controller Setup Time	Difference between the Arrival Time of FlowMod on OVS and Static Flow Push received Time on Controller (T2 - T1)

OVS Setup Time	Difference between the time stamp of Last packet on the firewall and the time of first offloaded packet directly on to the server connected port on the OVSd (T4-T3)
----------------	---

Table 5: Measurement Definitions

3. Measurements

Based on the above definitions of the Network delays, measurements are made and below are the sample plots. The observations drawn from the below results are listed in the below section.

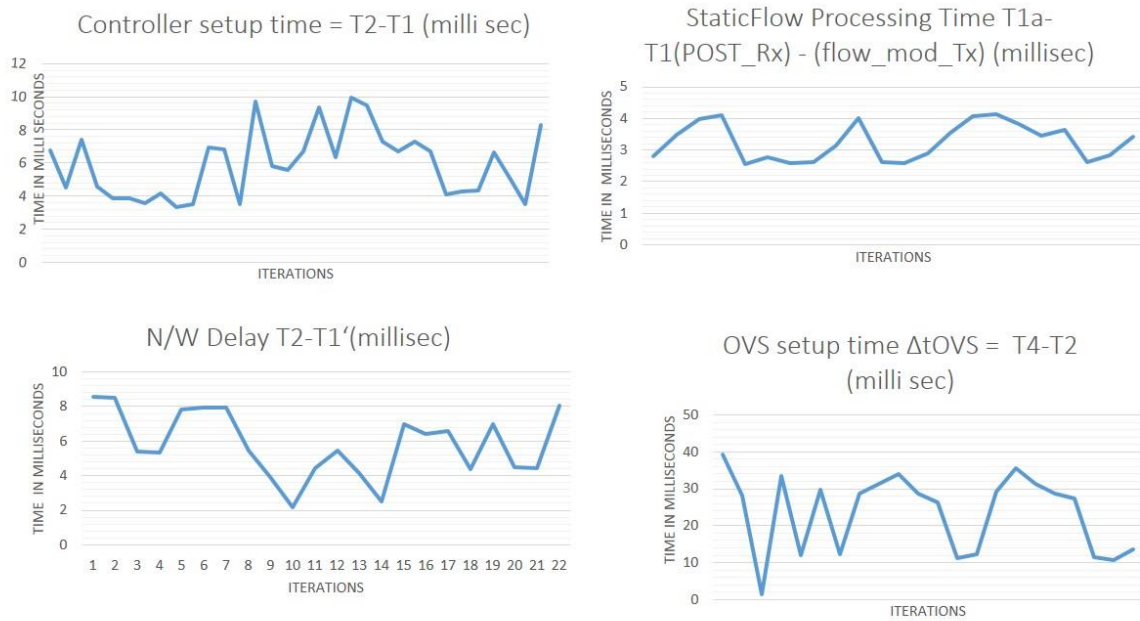


Figure 31: Network Delay Plots

4. Observations

From the above plots summarizing the controller setup time, static flow processing time, network delay and OVS setup time, the following observations are made:

- Static Flow Processing Time (FPT) varies in the range 2.5 to 4.1 milliseconds
- Controller Setup Time (CST) varies in the range 3.5 to 10.1 milliseconds
- Network Delay (ND) varies in the range 2 to 8 milliseconds
- In many cases, OVS Setup Time (OST) is ranging in two clusters; one ranging 10 to 14 milliseconds and the other ranging between 28 to 35 milliseconds; which is huge.
- $CST = FPT + ND$; with the inconsistent network delay, variation in CST is huge.

On investigation, we found following reasons for the above variations/ inconsistency:

- T2 and (T1, T1a) are on different machines. Clock synchronization is an issue.
- Delays due to virtualization are a major cause.

5. Xen Virtualization Delays

For understanding the delays caused by virtualization, we need to understand the following concepts: ^[20]

- pCPU – refers to physical CPU
 - It is a physical CPU core if hyper threading is unavailable
 - a logical CPU when hyper threading is available
- vCPU – refers to virtual CPU
 - a VMs virtual Processor
 - vCPU runs on a pCPU
 - *‘It is an execution context on a pCPU and, like a process, a vCPU can be in running state, ready state, wait state or wait_idle state’* [20]
- XEN Resource Scheduling

- In case of many VMs instantiated on one Xen Host, it is understood that all these VMs use the same physical machine's hardware. The XEN resource scheduling does the resource-sharing between VMs. Specifically, the CPU, hard disk and network card are involved.

Firstly, there is no relation between the number of vCPUs to be allocated to a pCPU or x pCPUs. Also there is no consideration for the above to ensure good performance. This is done only by estimating the work load on the virtual machines.

Also, on Xen virtualized environments, researchers made the below observations: [13]

- Processor sharing cause very unstable TCP/UDP throughputs
- Packet RTT abnormally varies – Unstable network performance

In this context, let us recall the physical resources and the virtual machine share of the available physical resources in Table 5 & Figure 5. The variance caused in ICMP RTT due to XEN virtualization is explained in 'The Impact of Virtualization on Network Performance of Amazon EC2 Data Center' [21]. Also, our OST measurements are justified in the explanations in 'Explaining Packet Delays under Virtualization'. [21]

Here in our case, the network latency is much more because of the following aspects:

- High number of vCPUs allocated which in-turn is huge work load on the Xen host
- Resource sharing and scheduling for data transfer on 20 virtual network interfaces on Xen host with 4 bridging instances of Open vSwitch

Chapter 7

Setup with Open Flow Hardware Switch & Distributed VM's

The earlier setup with all the VMs (Client, Server, AppServer & Firewall) on the same Xen host yielded highly variable output, which will be discussed in the next chapter.

Experimental setup with OpenFlow hardware switch is built. Also the VMs are distributed across multiple Xen hosts and the measurement point is introduced by tapping wire. The experimental setup is built with Pica8.

1. Experimental Setup

Figure 32 depicts experimental setup with OpenFlow hardware switch. The network connections and elements are explained below.

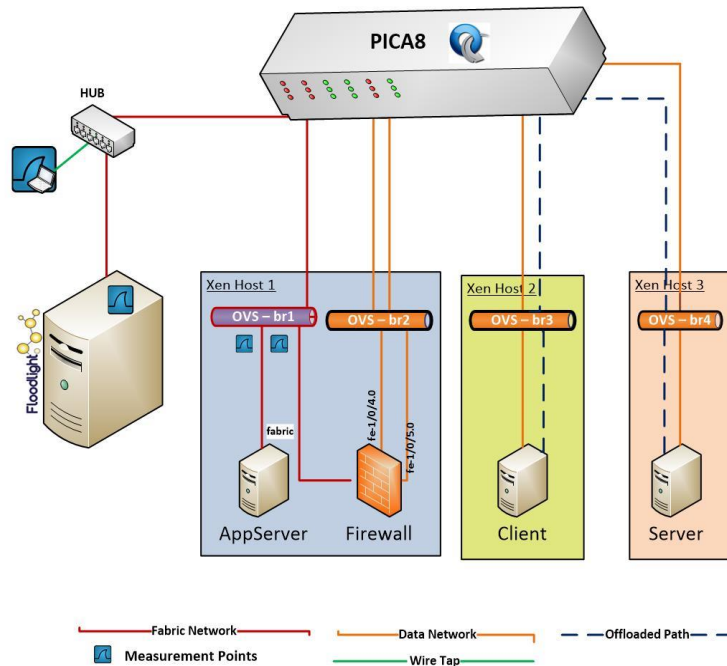


Figure 32: Experimental Setup with Hardware OpenFlow Switch

1. Client VM communication to Server VM will happen through the OpenFlow switch.
2. Client/Server VM to communicate with the other, will reach the OVS-br3/OVS-br4 respectively.
3. The physical NIC on Xen Host2 & Xen Host3 are bridged to the OVS-br3/OVS-br4 respectively.
4. Flows are written on OVS-br3/OVS-br4 such that the incoming traffic from the Client/Server is forwarded to the physical NIC, which in-turn is connected to the Pica8 OpenFlow hardware switch.

Also, flows are pushed on to the Pica8 such that the client to server traffic will not take the normal path, but will be forwarded to the firewall hosted PC.

5. Traffic ingresses the XenHost1 physical NIC1 which is bridged to OVS-br2 & flow rules written on the OVS-br2 will input the traffic to the firewall port.
6. Firewall DPIs traffic, identifies the application and sends out on the firewall out port. So, the traffic again arrives at the OVS-br2. Now, the traffic is forwarded to physical NIC2 (bridged to OVS-br2) of XenHost1.
7. Traffic arrives again at Pica8, on which the flow rules which were written already, directs the traffic to the Server connected port. Similarly the traffic coming from server follows the path directed by the flows written on the Pica8 and OVS-br2, OVS-br3, OVS-br4
8. App-Steering configuration on firewall, dictates to inform the AppServer about the on-going sessions as UDP messages.
9. In case of any offload rules already configured in AppServer, populates the static flow pushes with the information received in the form of UDP messages in conjunction with user application input.
10. These static flow pushes are sent to the Floodlight Controller, which in-turn sends out the OpenFlow Flow-Mod messages to the Pica8 OpenFlow Switch.
11. The traffic will be directed as per the new offload rules written on the Pica8 switch with the higher priority, dis-regarding the earlier flow rules.

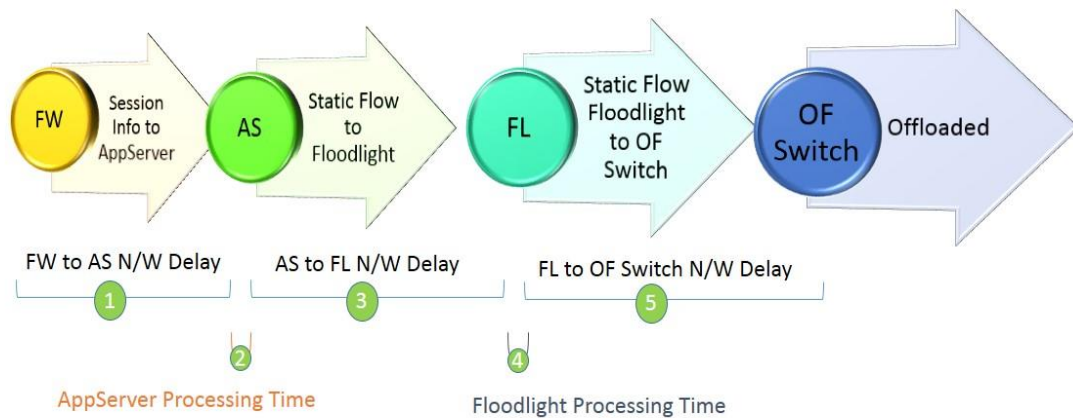
Chapter 8

Network Delays

1. What is minimum length of a session on firewall to benefit from offloading?

After a session is identified on the firewall, before it is offloaded, the following are the sequence of events:

Offloading – Events after session seen on FW



FW – Firewall ; AS – AppServer ; FL – Floodlight ; OF – OpenFlow ; N/W – Network

Figure 33: Offloading Events

In a broader sense, the above delays are to be characterized, analyzed and measured to answer the bigger question ‘*What is Minimum length of a session on firewall to benefit from offloading?*’

i. Firewall to AppServer Network Delay

- Firewall through the Fabric Interface, attached to OVS-br1, sends out a UDP message with the on-going session information on the firewall

```
set policy z1-z2 zone z1 z2 rule PERMIT-ALL match source-address Any
set policy z1-z2 zone z1 z2 rule PERMIT-ALL match destination-address Any
set policy z1-z2 zone z1 z2 rule PERMIT-ALL match service Any
set policy z1-z2 zone z1 z2 rule PERMIT-ALL match app-group Varmour::Any
set policy z1-z2 zone z1 z2 rule PERMIT-ALL action permit steering app-server
set profile app-steering app-server server-ip-address 10.0.2.20
set profile app-steering app-server server-port 30273
set profile app-steering app-server server-protocol udp
set profile app-steering app-server resend-interval 3
set profile app-steering app-server session-timeout 30
```

- Time taken for UDP messages to reach the AppServer from fabric interface.
Difference of timestamps between the UDP messages received on the firewall fabric connected port of OVS-br1 and AppServer connected port of OVS-br1 is the internal fabric network switching delay.

ii. AppServer Processing Delay

- AppServer after receiving the UDP message, processes the same and generates the Static Flow Push message to setup flows to the controller along with the user-input of the application.
- The time difference of the reception of the UDP message and the outgoing Static Flow Push (HTTP POST) is measured and this is the AppServer processing delay.

iii. AppServer to Floodlight Network Delay

- The AppServer sends out the generated Static Flow Push messages to the Floodlight Controller
- To characterize this one-way delay between AppServer and Floodlight Controller for relaying the Static Flow Pushes, the following methods are considered –
 - {POST – OK}
 - Static Flow Push messages are sent as HTTP POST.
 - HTTP OK messages are received in response.
 - Difference of timestamp between the HTTP POST and HTTP OK messages are calculated and can be termed as the AppServer to floodlight network delay.
 - But this would not be the exact network delay. This also has a component of delay of processing the HTTP POSTs and generating a HTTP OK in response.
 - RTT
 - ICMP RTT:
 1. Generally network delay are measured with ICMP RTT.

2. AppServer to Floodlight one-way network delay can be measured Rx-Tx of the ICMP messages

- TCP RTT:

1. This delay can be measured more realistically, by measuring the Rx-Tx of TCP ping messages with the load same as the Static Flow Pushes and the port number on which the floodlight is listening the Static Flows.

- The measurements are made at the wire tap PC from a hub. The hub introduced delay needs to be subtracted from the measurement.

iv. Floodlight Processing Time

- Floodlight Controller, generates and sends out the OpenFlow Flow-Mod messages to the Pica8 switch
- The difference in timestamp between the reception of HTTP POST message and outgoing OpenFlow Flow-Mod message on the Floodlight Controller is the Floodlight processing time.

v. Floodlight to OF Switch Network Delay

- The OpenFlow Flow-Mod's sent out of the controller reaches the Pica8 switch and the flow table is modified accordingly.
- At the wire-tap PC, difference between timestamps of OpenFlow Flow Mod messages received directly from the Floodlight and the OpenFlow Flow Mod's received on a different interface of the Wire Tap which is connected to the mirrored port of the Flow Mod receiving port on Pica8.

2. Determining Minimum Length of Session to Benefit from Offloading

The minimum length of the session on the firewall should be greater than or equal to the delays introduced by the offload process to get benefitted from the intelligent DPI bypass through offload rules. In the above chapters, we understood various events that all combine to result in the offload process. Also, we have attempted to determine how long does it for each event (network/processing delay) to occur. It is evident that, the sum of all the network and processing delays introduced by the offload process is the minimal length of the session on the firewall to get benefitted from the intelligent DPI bypass.

Chapter 9

Measurements

1. Measurement Points on the Experimental Setup

Let us quickly recall the measurement points on the experimental setup:

- Pica8 connected interface on the Floodlight Controller
- Firewall fabric & AppServer connected interfaces on OVS-br1
- Hub connected interface on WireTap PC
- AppServer interface in the firewall fabric network

2. Measurements

The time differences are measured on the above interfaces as per the delays defined in Chapter 8.

i. Firewall to AppServer network delay:

- The below graph explains Firewall to AppServer network delay
- The OVS switching delay is the prime component of this firewall to AppServer delay as both the Virtual Machines are present on the same physical machine

- From the below graph, it is evident that the delay mostly ranges between 0.2 & 0.3 milliseconds – This can also be accounted to the flow lookup and forwarding time matching the ingress traffic on the switch attached ports

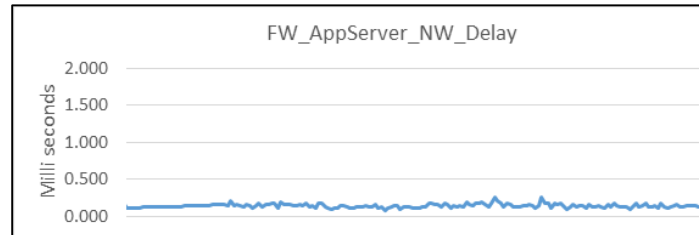


Figure 34: Firewall to AppServer Network Delay

ii. AppServer Processing Time:

- The below graph depicts the AppServer Processing Time
- After reception of the UDP message from the firewall with the session info, the AppServer takes typically 2 to 6 milliseconds to generate Static Flow pushes on to the controller linked to it
- Remember that this delay component has the delay component added up by the virtualization.

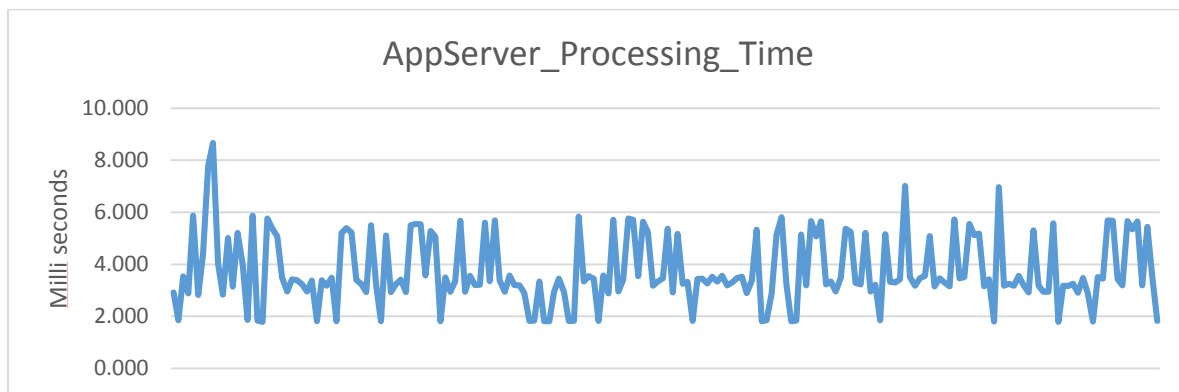


Figure 35: AppServer Processing Time

iii. Flow Setup Time:

- This delay is the sum of the Floodlight processing time and the network delay between the Floodlight and the OpenFlow switch.
- It is understood that the Floodlight Controller takes generally 2.2 to 4 milliseconds to process the static flow pushes.
- But often we can see the spike in the processing time of the static flow pushes
- It is found that these additional delays in the Floodlight static flow processing time is due to the Java garbage collection.
- Though the network delay is consistent, due to the variance in floodlight static flow processing time, this varies!

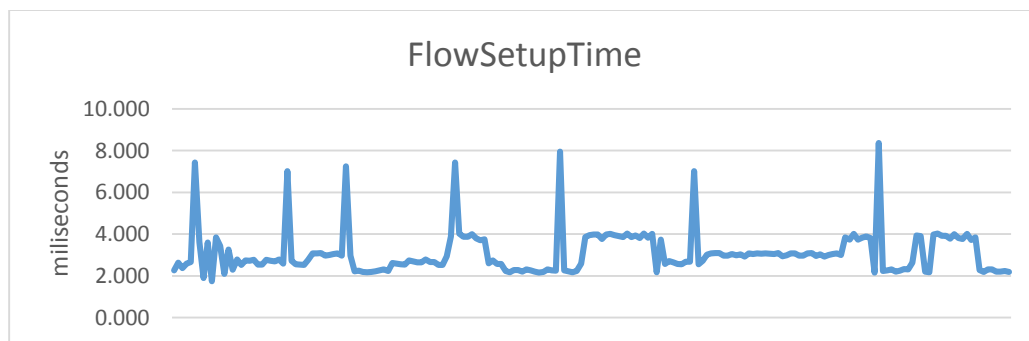


Figure 36: Sum of Floodlight Static Flow Processing Time and Floodlight to OVS network Delay

iv. Floodlight Processing Time:

- As discussed above, it is observed that the floodlight processing time varies between 2 to 3.9 milliseconds mostly.

- Please note that the measurements are made at different times and not a stretch of measurements done at a particular time.
- The additional delay in processing the static flow push and generating flow mod is due to the java garbage collection.

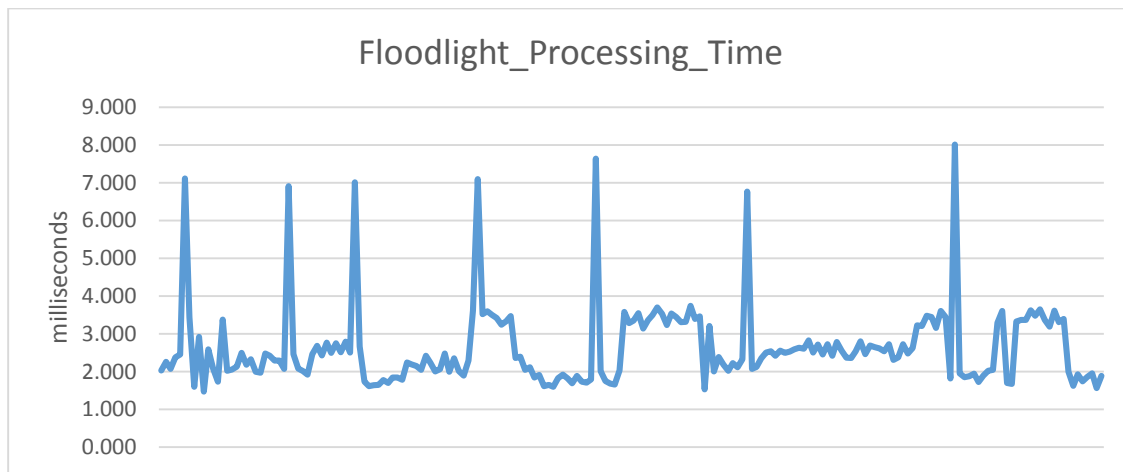


Figure 37: Floodlight Processing Delay

v. ***AppServer to Floodlight Network Delay:***

- The HTTP OK in response to the HTTP Static Flow Push sent by AppServer takes around 2.2 to 4.3 milliseconds on average
- This delay(w/ hub introduced delay) has actually the following components:
 - RTT of HTTP POST reaching the Floodlight from AppServer and the HTTP OK reaching the AppServer
 - HTTP OK response generation time by the Floodlight Controller
 - The HTTP OK response generation time has more variance in this measurement

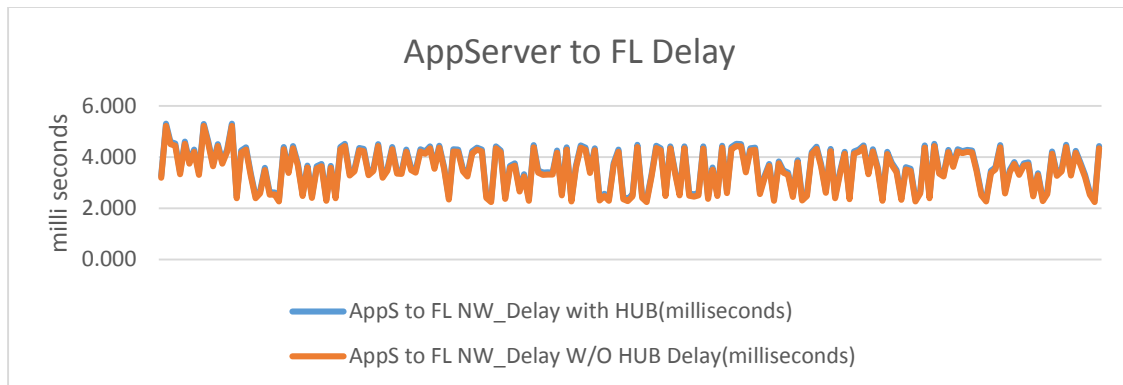


Figure 38: AppServer to Floodlight Network Delay

Chapter 10

GENI Experimentation

The experimental setup for Intelligent DPI offloading by Application Traffic Steering is built on the GENI [22] network resources and the network delays are measured.

1. GENI TOPOLOGY

The compute resources are reserved through the flack interface on the Protopeni [23] resources and the next section describes the below network topology.

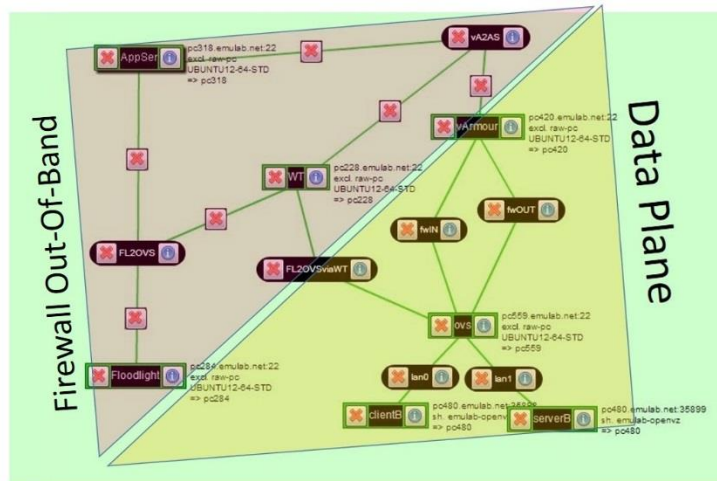


Figure 39: GENI Topology

2. Realization of Experimental Network Nodes on GENI

Let us discuss briefly how the topology is realized on the compute nodes:

- OpenFlow Switch – Open vSwitch is installed on a raw PC node; which is connected to the Client, Server and the Firewall in and out ports. A bridge is configured attaching the connected ports to the above mentioned network nodes. Flow rules are installed on the bridge to switch all the traffic passing from client to server to the firewall for deep packet inspection.
- AppServer – A raw PC node with Ubuntu operating system, is configured to be AppServer
- Client/Server – Linux VMs for passing traffic across.
- Floodlight – A raw Ubuntu PC on which the Floodlight Controller is running.
- WireTap – The measurement point on the topology.

Chapter 11

Results

As discussed in the Chapter 9, the experimentation is done and Firewall to AppServer network delay, AppServer processing time, AppServer to Floodlight network delay, Floodlight processing time & Floodlight to OpenFlow switch network delay are found. The below graphical representations summarize the above delays:

i. Firewall to AppServer Network Delay

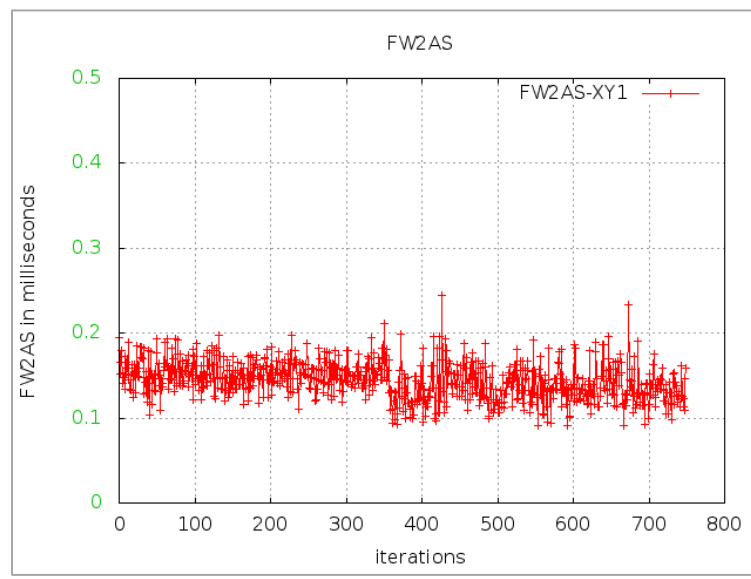


Figure 40: Firewall to AppServer Network Delay

- Graphical representation of the same is made in Figure 40.
- Firewall and AppServer are connected through a measurement point node, WireTap.
- UDP traffic from the firewall to AppServer flows through WireTap.
- Additional delay caused by traffic flowing through WireTap is not measured for the following reasons:
 - Ideally the Firewall/AppServer/Controller cannot be directly connected nodes
 - All of them will be connected through a switching element. Implies they would be one hop away from each.
 - In the above case, say the delay from the firewall to the switching element and switching element to the AppServer is visualized as the delay from firewall to WireTap and then from WireTap to the AppServer.
- As the Firewall is a Virtual Machine on the KVM hypervisor on a Raw PC
 - This delay component has the KVM bridge delay component too.
 - The overall Firewall to AppServer network delay constitute 0.125 to 0.175 millisecond range. Average delay is 0.169 milliseconds on a set of 7000 measurement points.
 - Please note that this is not the Round Trip Time. It is just one way UDP message AppServer reaching time from the Firewall.

ii. AppServer to Floodlight Network Delay:

- Please refer to the definition and measurement options in section 1 of Chapter 8. The three possibilities are plotted in As discussed in the above mentioned section, network delay is not the only component in the {POST-OK} method. So the RTT methods are

employed to find the one way network delay between the AppServer and Floodlight Controller. For further considerations TCP based measurement is used.

- On the topology built on raw physical nodes on GENI, AppServer to Floodlight network delay ranges from 0.15 to 0.175 milliseconds. This is the ICMP one-way network delay. The same is plotted in Figure 10.3(a). Average delay is 0.1543 milliseconds
- On the WireTap, the difference between the timestamp's of HTTP POST message from the AppServer and HTTP OK sent in response for the HTTP POST are plotted in Figure 10(b). Average delay is 5.191 milliseconds
- The AppServer to Floodlight one way network delay is measured by generating TCP packets with same frame size as the Static Flow Push messages (278 bytes in this experiment) and sent with destination port 80. The same is plotted in Figure 10(c). The average delay is 0.17 milliseconds

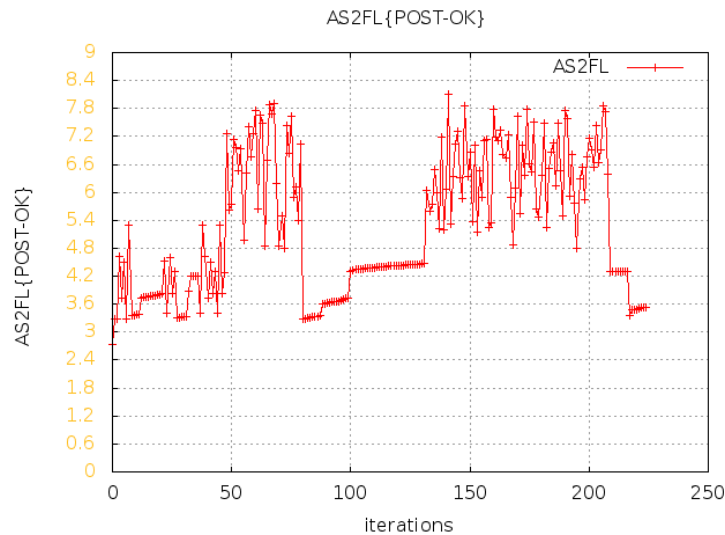


Figure 41: AppServer to Floodlight Network Delay {POST-OK} (b)

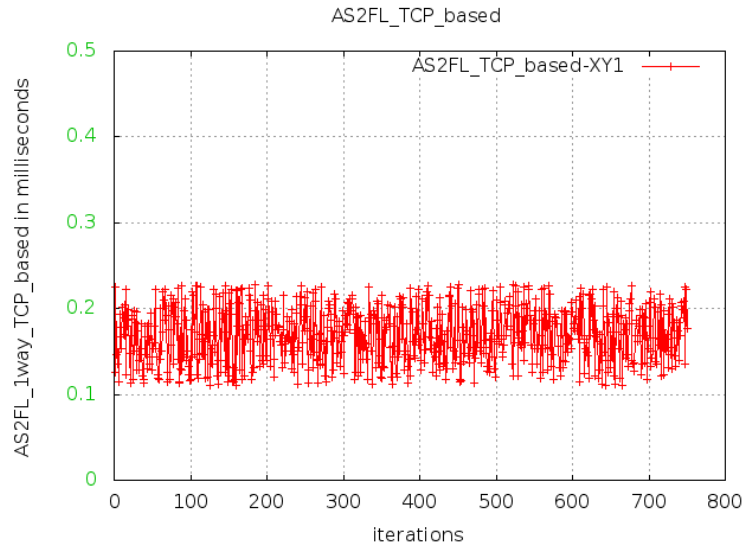


Figure 42: AppServer to Floodlight Network Delay TCP RTT (c)

iii. AppServer Processing Time:

- It takes 4 to 6 milliseconds to typically process the UDP session info from the firewall and generate static flow pushes on to the Floodlight Controller
- The spikes in the processing delay are due to java garbage collection.

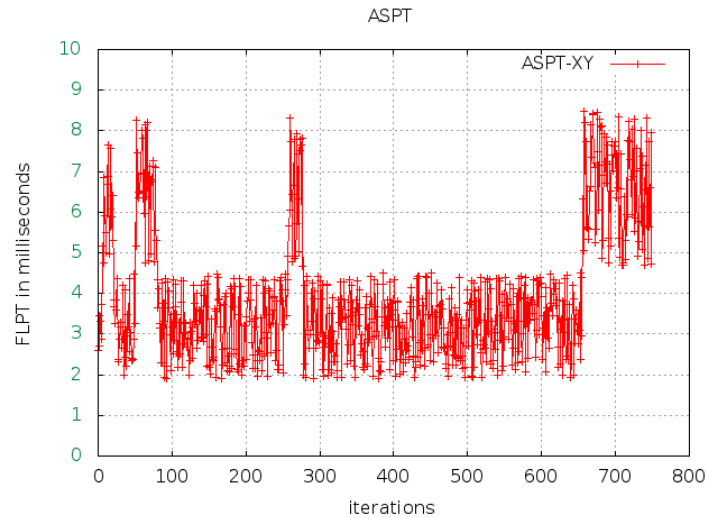


Figure 43: AppServer Processing Time

iv. Floodlight to OVS Network Delay:

- On the GENI emulation topology shown in Figure10, the average one-way network delay between the Floodlight Controller and OVS is 0.115 milliseconds
- After the Flow Mod reaching the Open vSwitch, it processes the Flow Mod and writes the same to the flow table.

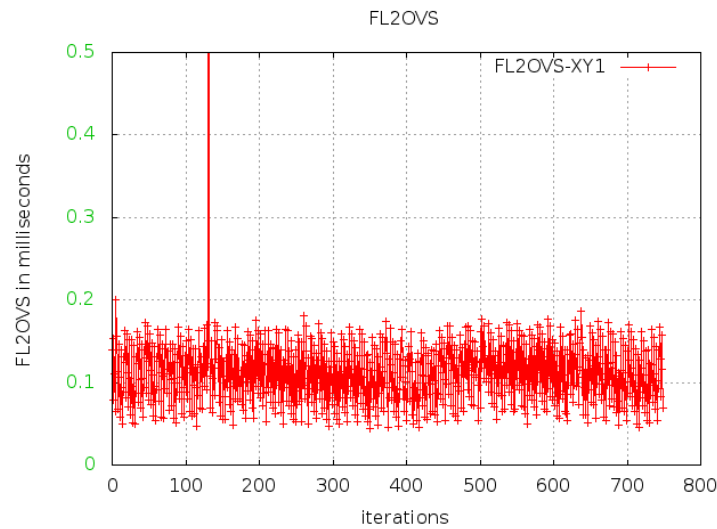


Figure 44: Floodlight to OVS Network Delay

v. Floodlight Processing Time:

- It takes 4.5 to 6 milliseconds generally to process a static flow push and generate a flow mod to the switch it is connected with.
- The spikes in the processing delay are due to the Java garbage collection.
- It takes relatively less time for the Floodlight Controller to generate a reactive flow rule from its already learnt topology.

- On average Floodlight took 4.94ms to process the Static Flow Pushes and convert to OpenFlow Flow Mod's

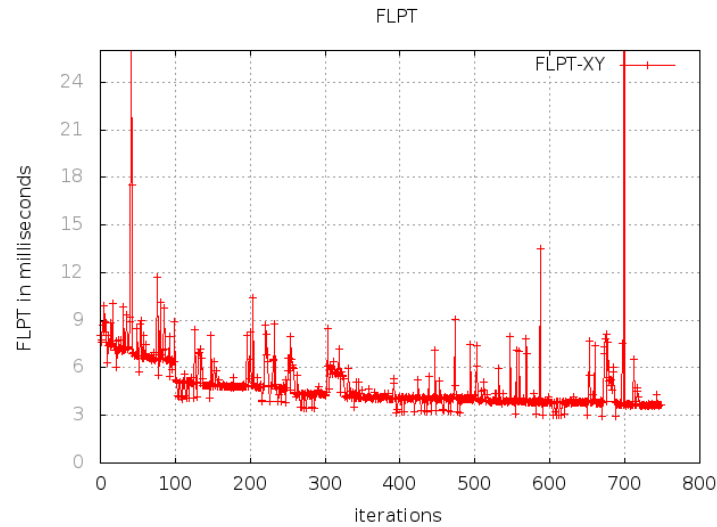


Figure 45: Floodlight Processing Time

All the above mentioned delays sum up to the minimum length of the session on the firewall to benefit from the intelligent offloading as discussed in the section ‘Determining the Minimum length of session to benefit from Offloading’.

The below graph represents the total time it takes to push the new offload rules to the OpenFlow switching element after identifying the application on the firewall.

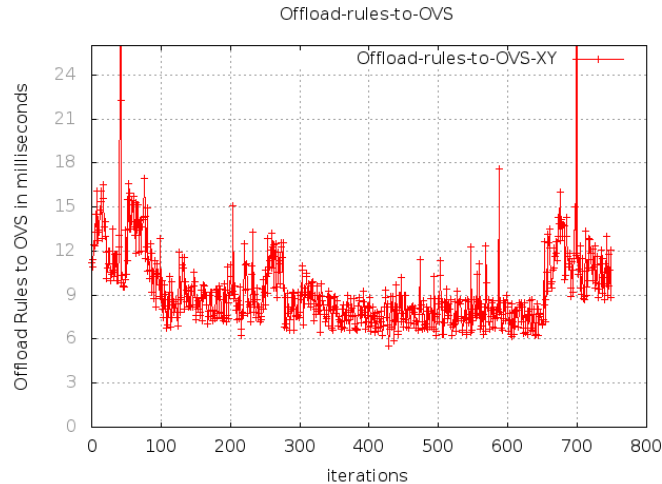


Figure 46: Offload rules to OVS

vi. How long does it take to identify SCP application on firewall?

Experiments are conducted to find out the average time it takes to identify the SCP application traffic on the firewall.

- The below graph depicts the SCP application identification times for number of iterations.
- It is found that the SCP application identification time ranges between 20.1 milliseconds and 21.8 milliseconds.
- Also, the time difference between the iterations is greater than the session expiration time on the firewall.
- Hence, for every iteration the firewall identifies the application and maintains the new state information.

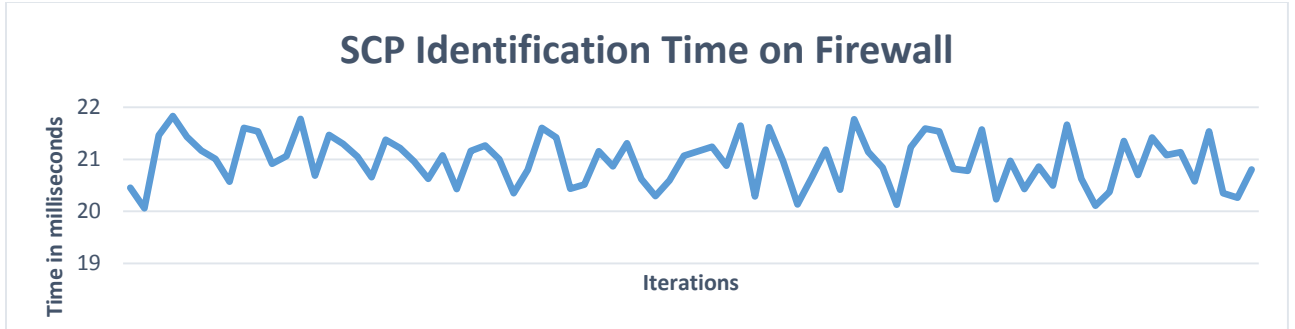


Figure 47: SCP Application Identification Time on Firewall

The below graphs are the representations of all the network delay's it takes to offload the fast path rules to the OpenFlow switch(OVS in our case)

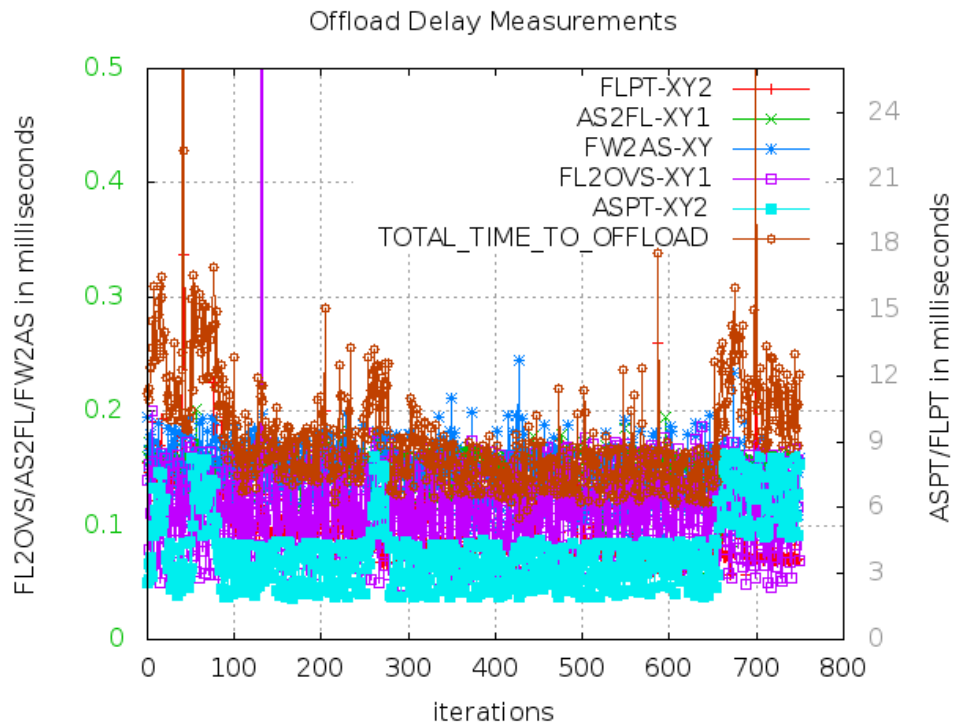


Figure 48: Offload Delay Measurements (a)

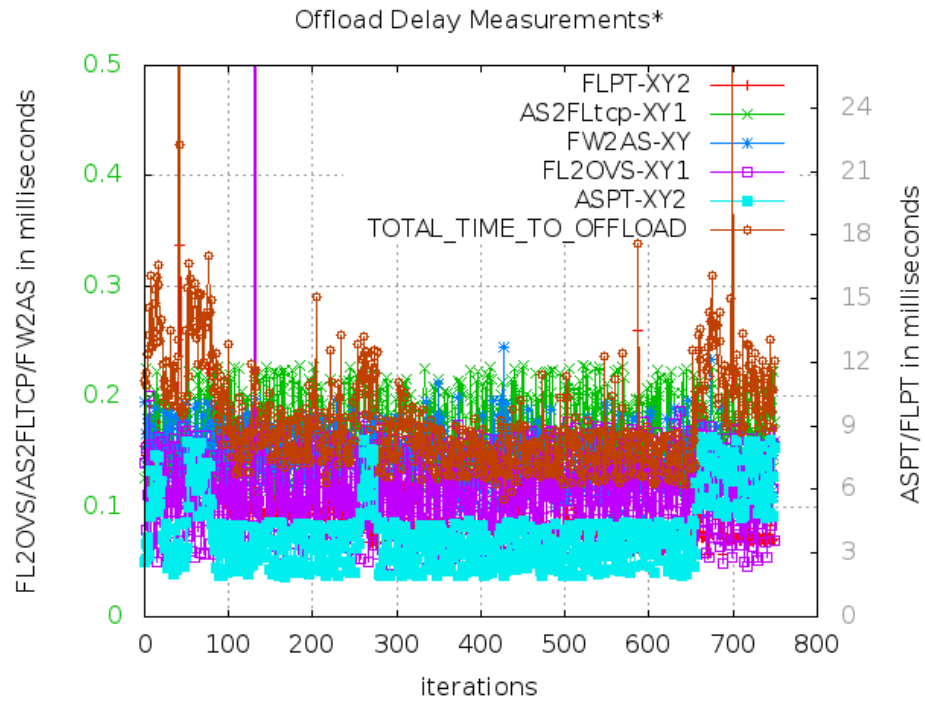


Figure 49: Offload Delay Measurements (b)

Chapter 12

Conclusion

From the results yielded from the experiment to find out the time it takes for the new offload rules to reach the OpenFlow switch, the following conclusions can be made:

For the virtualized setup, the following conclusions can be drawn:

- Mean time and the 99th percentile time for the new offload flows to reach OpenFlow switch for the fast path are calculated.
- It takes 22.11 milliseconds on average to push the new offload rules to the OpenFlow switch for the fast path and the 99th percentile is 32.7435 milliseconds. The split of all the delays involved are tabulated and graphed below.
- The Floodlight Processing Time takes 83.68% of the 99th percentile of total time to offload and 77.55% of the mean total time to offload.

Please note that the following terms are used further:

- FLPT – Floodlight Processing Time;
- ASPT – AppServer Processing Time;

- AS2FL – AppServer to Floodlight network delay;
- FL2OVS – Floodlight Controller to Open vSwitch network delay;
- FW2AS – Firewall to AppServer network delay

DELAY	FLPT	ASPT	FL2OVS	AS2FL	FW2AS
99th Percentile	27.4	4.497	0.27	0.394	0.1825
Mean	17.14892	3.873931	0.562724	0.373621	0.152914

Table 6: Mean & 99th Percentile of Total Time to Offload on Virtualized Setup

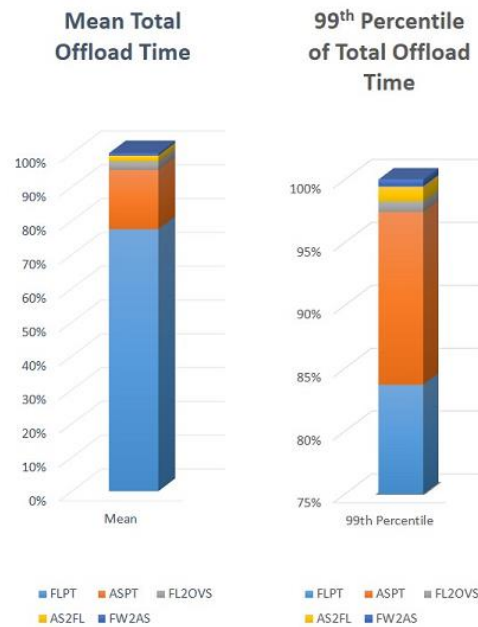


Figure 50: Mean & 99th Percentile - Time to Offload - Virtualized Setup

For the hardware PC setup, the following conclusions can be drawn:

- Mean time and the 99th percentile time for the new offload flows to reach the OpenFlow switch for the fast path are calculated.
- It takes 9.223 milliseconds on average to push the new offload rules to the OpenFlow switch for the fast path and the 99th percentile is 16.066 milliseconds. The split of all the delays involved are tabulated and graphed below.

Delay	FLPT	ASPT	FL2OVS	AS2FL	FW2AS
Mean	4.903049	3.909264	0.112401	0.154378	0.143981
99th Percentile	8.84	6.884523	0.159	0.1335	0.0495

Table 7: Mean & 99th Percentile of Total Time to Offload on Physical PC Setup

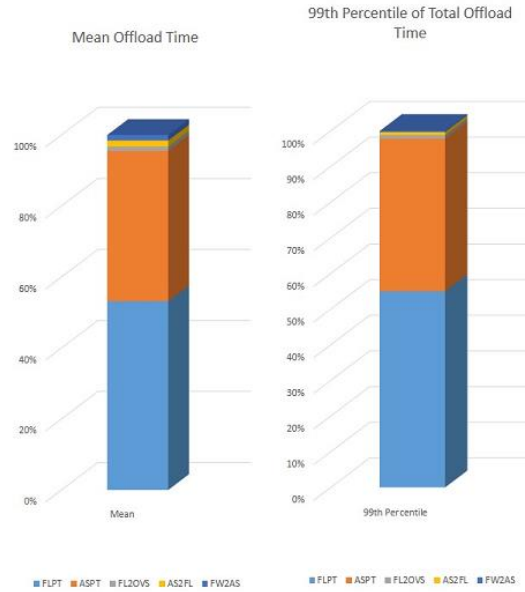


Figure 51: Mean & 99th Percentile - Total Time to Offload - Physical PC Setup

The following observations provide insight on which measurements can be generalized out of the arrived results:

- Network Delays have minimal variance on comparison to the processing delay's on iterations & virtualization causes additional processing delays due to the processor/resource sharing between the VMs.
- On the virtualized setup, Floodlight processing time takes 55.023% of the 99th 1percentile of Total Time to offload and 53.16% of the mean total time to offload whereas it is a much different scenario when the physical PC experimental setup results are considered.

- In majority of the cases, network delays will be in the same order as arrived results.
The setup is built considering the network architecture of firewall placement and most probable placements (number of hops) of AppServer, OpenFlow controller.
- Consideration on the type of the machine (physical or virtual) and virtual to physical resource mapping (in case of virtual machines) is required for measurement of Floodlight & AppServer processing times. We have mentioned the resources specifications of the experimental PCs we have used. Measured Floodlight & AppServer processing times may be considered accordingly.

Chapter 13

Future Work

The following are the important further aspects to study in regard to the current work:

- Determining CPU and Memory Utilization savings on the firewall by offloading safe traffic
- Understand the variation in response time of OpenFlow switch flow match when the flow rule has minimal match criteria & the flow rule using maximum tuples.
- AppServer enhancements like:
 - Capability to configure offload rule's to identify different streams of the same application traffic between two nodes
 - Integrate AppServer with the OpenFlow controller
 - Switch based profile selection menu instead of existing drop down selection
 - Use Case – AppServer attached to multiple controllers & each controller having multiple OpenFlow switches attached.

References

1. CISCO systems, "Deploying Firewalls Throughout Your Organization." Accessed January 8, 2014.
http://www.cisco.com/en/US/prod/collateral/vpndevc/ps5708/ps5710/ps1018/prod_white_paper0900aecd8057f042.pdf.
2. Luo, Shengmei , Zhaoji Lin, and Xiaohua Chen. ZTE Corp., Shenzhen, China, "Virtualization security for cloud computing service." Last modified December 14, 2011. Accessed January 8, 2014.
http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6138516&url=http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6138516.
3. The Open Networking Foundation, "Software-Defined Networking (SDN) Definition." Accessed January 8, 2014. <https://www.opennetworking.org/sdn-resources/sdn-definition>.
4. The Open Networking Foundation, "OpenFlow Specification 1.4." Last modified October 14, 2013. Accessed January 8, 2014.
<https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>.
5. Curran, Dominic. Citrix Systems Inc, "Open vSwitch & OpenFlow in XCP & XenServer." Last modified November 30, 2012. Accessed January 8, 2014.http://www.slideshare.net/xen_com_mgr/under-the-hood-open-vswitch-openflow-in-xcp-xenserver.

6. vARMOUR Networks Inc, "Software Defined Security - SDSec." Accessed January 8, 2014.<http://www.varmour.com/technology.html>.
7. SANS Institute, "Understanding Intrusion Detection Systems." Accessed January 8, 2014.<http://www.sans.org/reading-room/whitepapers/detection/understanding-intrusion-detection-systems-337?show=understanding-intrusion-detection-systems-337&cat=detection>.
8. Eli Dart, Lauren Rotman, Brian Tierney, Jason Zurawski, and Mary Hester. "The Science DMZ: A Network Design Pattern for Data-Intensive Science." *SuperComputing 13*.
9. Triwedi, Uday. "A self-learning stateful application identification method for Deep Packet Inspection." *International Conference on Computing Technology and Information Management (ICCM)* : 416-421.

<http://ieeexplore.ieee.org.ezproxy.lib.uh.edu/stamp/stamp.jsp?tp=&arnumber=6268534>(accessed January 8, 2014).1
10. Shaw, Chris. "Security Focus Focus of new EEMBC benchmark suite." Accessed January 8, 2014. <http://www.newelectronics.co.uk/electronics-news/security-focus-of-new-eembc-benchmark-suite/26872/>.
11. Khalife, Jawad M, Jesus Díaz-Verdejo, and Amjad Hajjar. "Performance of OpenDPI in Identifying Sampled Network Traffic." *Academy Publisher*. no. 1 (2013): 71-81.

<http://ojs.academypublisher.com/index.php/jnw/article/view/8396/0> (accessed January 8, 2014).
12. Wikipedia, "XEN." Accessed January 8, 2014.<http://en.wikipedia.org/wiki/Xen>.
13. Wang, Guohui, and T.S. Eugene Ng. "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center." *INFOCOM'10*. : 1163-1171.

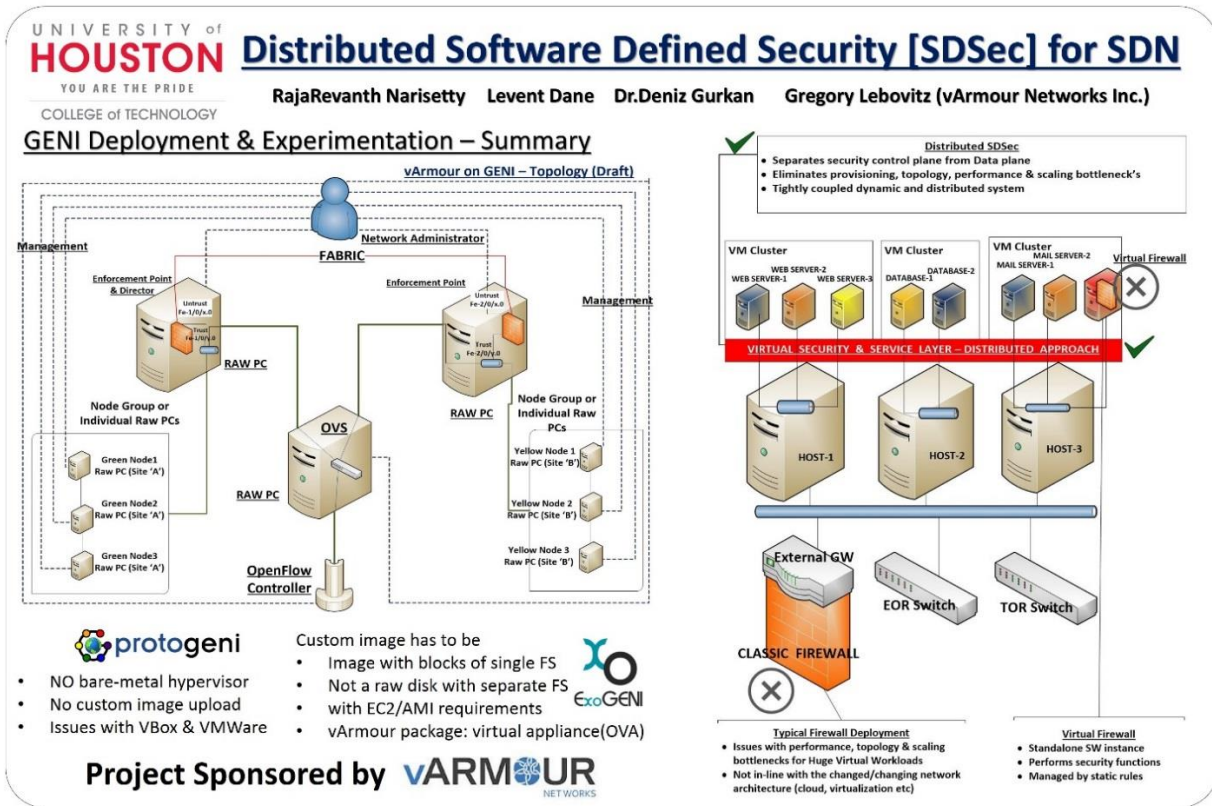
- <http://www.cs.rice.edu/~eugeneng/papers/INFOCOM10-ec2.pdf> (accessed January 8, 2014).
14. McKeown, Nick, Hari Balakrishnan, Scott Shenker, Guru Parulkar, Tom Anderson, and Larry Peterson. "OpenFlow: Enabling Innovation in Campus Networks." <http://archive.openflow.org/documents/openflow-wp-latest.pdf> (accessed January 8, 2014).
15. "Open virtual Switch." Accessed January 8, 2014. openvswitch.org.
16. BigSwitch Networks Inc, "Static Flow Pusher API." Accessed January 8, 2014. [http://www.openflowhub.org/display/floodlightcontroller/StaticFlow Pusher](http://www.openflowhub.org/display/floodlightcontroller/StaticFlow+Pusher)
17. cURL, "cURL Manual Page." Accessed January 8, 2014. <http://curl.haxx.se/docs/manpage.html>.
18. Marist College, "What is Avior?." Accessed January 8, 2014. <http://openflow.marist.edu/avior>.
19. TCPDUMP/LIBPCAP public repository, "TCPDUMP & LIBPCAP." Accessed January 8, 2014. <http://www.tcpdump.org/>.
20. Virtual Byte | virtualization & More, "Some thoughts about pCPU/vCPU." Accessed January 8, 2014. <http://virtualbyte.wordpress.com/2010/12/01/some-thoughts-about-pcpuvcpu/>.
21. Whiteaker, Jon, Fabian Schneider, and Renata Teixeira. "Explaining packet delays under virtualization." *ACM SIGCOMM Computer Communication Review*. no. 1 (2011): 38-44. <http://dl.acm.org/citation.cfm?id=1925867>(accessed January 8, 2014).
22. GENI, "GENI - Exploring Networks of future." Accessed January 8, 2014. <http://www.geni.net>.

23. Protogeni, "WikiStart - Protogeni." Accessed January 8, 2014.

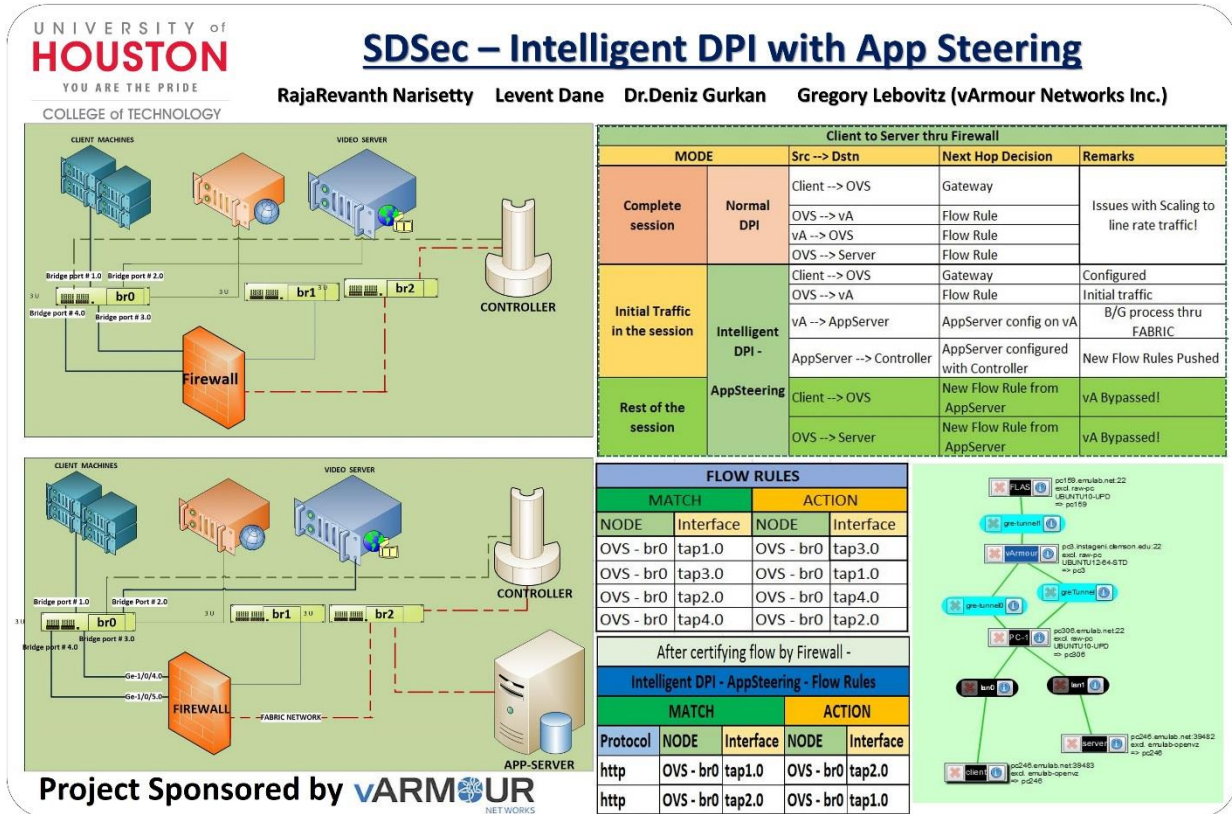
<http://www.protogeni.net/wiki>.

Appendix

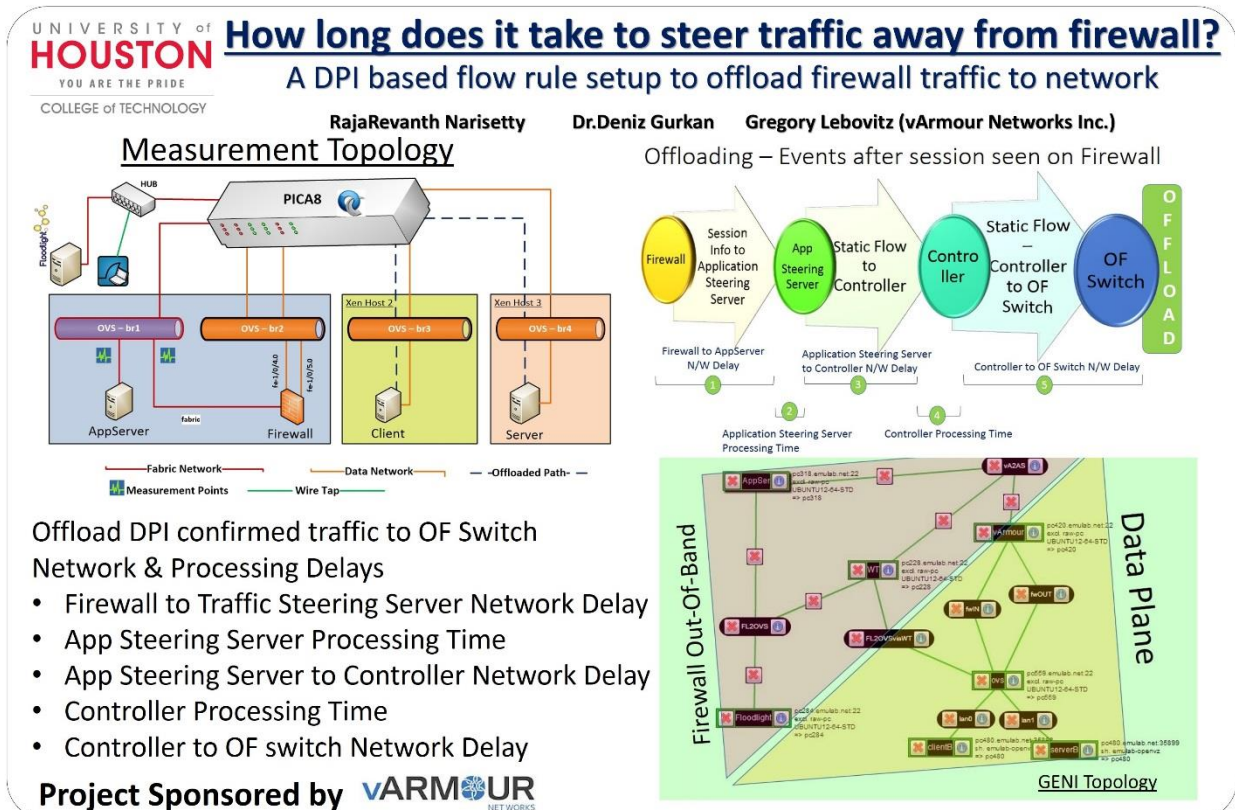
1. GEC 16 POSTER



2. GEC 17 POSTER



3. GEC 18 POSTER



Bibliography

- I. Describes the updated OpenFlow switch specifications added to the already existing features. The latest updates of the OpenFlow protocol messages are listed.

Open Networking Foundation

<https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>

- II. Explains the need for programmable switches and networks. Describes the OpenFlow switch and Controller with mention of the new flow rule based switching capabilities. The flow tables & flow table lookup for switching the traffic are explained.

OpenFlow: Enabling Innovation in Campus Networks / March 14, 2008

Nick McKeown, Stanford University; Tom Anderson, University of Washington

Hari Balakrishnan, MIT ;Guru Parulkar, Stanford University

Larry Peterson, Princeton University; Jennifer Rexford, Princeton University

Scott Shenker, University of California, Berkeley; Jonathan Turner, Washington

University in St. Louis

<http://archive.openflow.org/documents/openflow-wp-latest.pdf>

- III. Extensive mention about the existing/updated configuration and operational limitations of the HP ProCurve switch. Detailing on the VLAN virtualization and Aggregation Modes is available.

<http://bizsupport2.austin.hp.com/bc/docs/support/SupportManual/c03512348/c03512348.pdf>

- IV. Using the property that many of the application flows have correlated flows and they can be identified by existing DPI methods, a novel approach is proposed. Keeping information of identified flows during an unidentified flow and when enough records are found, correlation of the unidentified flow with identified flow is made which results in concluding the unidentified flow being part of an identified flow application. This method identifies application flows with the help of correlation flows and statistical analysis. Also discussion on existing DPI methods like port based, pattern based, flow statistics and flow behavior based.

A Self-learning Stateful Application Identification – Method for Deep Packet Inspection

Uday Trivedi

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6268534>

- V. Comprehensively discussed the need for Science DMZ. Aspects like Architecture, design, implementation, security, performance tuning, application selection of the science DMZ has been shared. Use cases of successful implementation of the Science DMZ at the University of Colorado at Boulder, The Pennsylvania State University & Virginia Tech Transportation Institute, The National Oceanic and Atmospheric Administration and National Energy Research Scientific Computing Center. Abilities and advantages of

OpenFlow to dynamically modify security policies for large flows between trusted zones are discussed.

The Science DMZ: A Network Design Pattern for Data-Intensive Science

Eli Dart, Lauren Rotman, Brian Tierney, Mary Hester, Jason Zurawski

http://www.es.net/assets/pubs_presos/sc13sciDMZ-final.pdf

- VI. Leveraging capabilities of Software Defined Networking with OpenFlow protocol new architectural proposals have been examined on SC11 SCinet Research Sandbox demonstrator with in the science-network. Explained how ECSEL (using ESNet's OSCAR service) benefits a campus implementing a data transfer service can leverage OpenFlow for end-to-end connectivity and flexibility. Discussed implementation & challenges of complex use cases like having multiple science centers, OpenFlow/SDN science DMZ architecture, and Dynamic Science collaborations.

Software-defined networking for big-data science – Architectural models from campus to

WAN : Inder Monga, Eric Pouyoul, Chin Guok, Energy Sciences Network

http://www.es.net/assets/pubs_presos/ESnet-SRS-SC12-paper-camera-ready.pdf

- VII. Network traffic study is conducted at 10 data centers of three variants like University, Enterprise and Cloud. SNMP polls, Network Topology and packet traces are collected for analysis. Interesting findings include '*Most flows in the data centers are small in size ($\leq 10KB$), a significant fraction of which last under a few hundreds of milliseconds, and the number of active flows per second is under 10,000 per rack across all data centers*'. Application Communication patterns including Flow-level communication characteristics

like Flow Sizes, Flow lengths, packet sizes, Number of active flows, Flow inter-arrival times' are observed.

Network Traffic Characteristics of Data Centers in the Wild

*Theophilus Benson**, *Aditya Akella** and *David A. Maltz†*

**University of Wisconsin–Madison †Microsoft Research–Redmond*

<http://bnrg.cs.berkeley.edu/~randy/Courses/CS294.S13/3.4.pdf>

- VIII. Developed tool, cbench to quantify OpenFlow controller performance like number of flow setups per second a controller can handle. The measurements can made in throughput mode and latency mode.

On Controller Performance in Software-Defined Networks

Amin Tootoonchian University of Toronto/ICSI

Sergey Gorbunov University of Toronto

Yashar Ganjali University of Toronto

Martin Casado Nicira Networks

Rob Sherwood Big Switch Networks

<http://dl.acm.org/citation.cfm?id=2228297>

- IX. OFLOPS, a tool to measure performance and test the capabilities of an OpenFlow software and hardware switches. Tests to find out packet processing time, Flow table update rate, OpenFlow monitoring capabilities, impact of interaction between OpenFlow operations are made.

OFLOPS: An Open Framework for OpenFlow Switch Evaluation

Charalampos Rotsos, Nadi Sarrar, Steve Uhlig, Rob Sherwood and Andrew W. Moore

<http://www.net.t-labs.tu-berlin.de/papers/RSUSM-OOFOFSE-12.pdf>

- X. Switching times are measured for hardware and software OpenFlow implementations. Based on these a model has been proposed to measure the blocking probability and forwarding speed of an OpenFlow switch with OF Controller. Packet sojourn time and probability of lost packets has been measured with this model. Patterns of Forwarding delay of OpenFlow switch (hardware and software) with payload is found out.

Modeling and Performance Evaluation of an OpenFlow Architecture

Michael Jarschel, Simon Oechsner, Daniel Schlosser, Rastin Pries, Sebastian Goll, Phuoc Tran-Gia

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6038457>

- XI. A common testing Methodology has been proposed for performance measurement of Deep Packet Inspection of Firewalls. DPI Benchmarking is done at different scalability levels (number of flows) of network appliances.

Performance and Measurement of DPI Technologies

Proposal for a Common Testing Methodology: DPIBench

White Paper v1.3 | Jeff Caldwell, SonicWALL Markus Levy, EEMBC

- XII. pS, performance tool kit, is a framework of performance tools, middleware, visualizations and alarms .This tool set measures the network performance with the parameters like jitter, achievable bandwidth, latency using tools like NPAD & NDT, OWAMP and BWCTL

Firewall Port Recommendations for the pS Performance Toolkit

Internet2 Performance Working Group, March 2013

Edited by J. Zurawski (Internet2), A. Brown (Internet2), A. Lake (ESnet), K. Miller (The Pennsylvania State University), M. Swamy (Indiana University), B. Tierney (ESnet), and M. Zekauskas (Internet2)

<http://psps.perfsonar.net/toolkit/20130628-Firewall-PerfWG.pdf>

- XIII. Impact of virtualization on RTT (round trip time) is measured on Linux-VServer and XEN. Results are Round trip time is dissected and analyzed. It is found out that heavy network traffic from competing virtual machines can introduce significant delay to RTT measurements and most delay is introduced while sending packets (as opposed to receiving packets).

Explaining Packet Delays under Virtualization

Jon Whiteaker, Fabian Schneider, Renata Teixeira

UPMC Sorbonne Universities and CNRS

ACM SIGCOMM Computer Communication Review

<http://www.net.t-labs.tu-berlin.de/~fabian/papers/ccrJan11whiteaker-paper.pdf>

- XIV. Measurements to characterize the impact of virtualization on the network in Amazon Elastic Cloud Computing. Processor sharing, packet delay, TCP/UDP throughput and packet loss are measured on virtualized environments. Some of the worthwhile observations include that the virtual machines receive processor share of 40% - 50%, this results in very unstable TCP/UDP throughput, abnormal packet delays due to long

queuing delays at the driver domains of the virtualized machines. Also abnormally unstable network performance can dramatically skew the results of certain network performance measurement techniques.

The Impact of Virtualization on Network Performance of Amazon EC2 Data Center

Guohui Wang T. S. Eugene Ng

Dept. of Computer Science, Rice University

<http://www.cs.rice.edu/~eugeneng/papers/INFOCOM10-ec2.pdf>

- XV. Proposes a framework to solve current security vulnerabilities and threats (like attack between VMs or attack between VM and VMM, VM escape, VM controlled by host machine, Denial of Service, VM Sprawl) in two blocks – one being virtual security and the other being virtualization security management mechanisms. ^[15]

“Virtualization security for cloud computing service”, International Conference on

Cloud and Service Computing, Shengmei Luo, Zhaoji Lin, Xiaohua Chen ZTE Corporation

and Zhuolin Yang, Jianyong Chen, Shenzhen University, Shenzhen, China

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6138516>

INDEX

Application Layer, 13

AppServer, 30

AppServer Processing Time, 69

AppServer to Floodlight Network Delay, 69

AVIOR, 38

behavioral study, 25

Clock synchronization, 51

Configuration Management, 17

Control Layer, 13

Controller, 14

Controller setup time, 50

Controller to OVS Network Delay, 50

CURL, 38

Data Centers, 11

data plane, 13

Deep Packet Inspection, 20

Deep Packet Inspection throughput, 19

Distributed Firewalls, 12

domain 0, 30

DPIBench, 25

Firewall, 30

Firewall Bottleneck, 19

Firewall Placement, **11**

Firewall to AppServer Network Delay, 69

Floodlight, 32

Floodlight Processing Time, 69

Floodlight to OpenFlow Switch, 69

flow tables, 32

flow_mod, 16

forwarding plane, 13

GENI, 67	Pica8, 32
guest domains, 30	PICA8, 32
Host Intrusion Detection System (HIDS), 21	Protogeni, 67
HTTP OK, 60	Science DMZ, 21
HTTP POSTs, 60	SCP, 42
intelligent, 73	SDSec, 33
Intrusion Detection/Prevention System, 20	Shallow Packet Inspection, 20
Intrusion Prevention System throughput, 19	Software Defined Security (SDSec), 17
Network Intrusion Detection System, 21	Static Flow Processing Time, 50
Network Node Intrusion Detection System (NNIDS), 21	static flow push, 42
Open vSwitch, 32	Tcpdump, 40
OpenDPI, 25	<i>the time to offload</i> , 50
OpenFlow, 14	vARMOUR, 33
OpenFlow dissector Wireshark, 40	vCPU, 52
OVS Setup Time, 50	Virtual Firewalls, 12
Packet Out, 16	virtualization, 30
Packet_In, 16	wire tap, 60
pCPU, 52	XEN, 30
pCPUs, 30, 31	XEN Resource Scheduling, 52