UNIVERSITY OF NAIROBI

SCHOOL OF COMPUTING AND INFORMATICS

DISTRIBUTED DENIAL OF SEVICE MITIGATION IN SOFTWARE DEFINED NETWORKING

BY GODFREY NENGO KIHAMBA

P53/85511/2016

SUPERVISOR

DR. ELISHA ABADE

A RESEARCH PROJECT REPORT SUBMITTED TO THE SCHOOL OF COMPUTING AND INFORMATICS IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF MASTER OF SCIENCE IN DISTRIBUTED COMPUTING TECHNOLOGY OF THE UNIVERSITY OF NAIROBI.

JULY 2019.

DECLARATION

I certify that this research project report to the best of my knowledge, is my original authorial work except as acknowledged therein.

Signature: ------------------------------------ Date: --

Godfrey Nengo Kihamba (P53/85511/2016)

This research report has been submitted in partial fulfilment of the requirements for the Degree of Masters of Science in Distributed Computing Technology at the University of Nairobi with my approval as the University supervisor.

Signature: ------------------------------------ Date: --

Dr. Elisha Abade.

DEDICATION

To my beloved parents for their dedicated and unending support in my education pursuits.

To my dear brother and sisters for their moral support during this long pursuit of my Master’s degree.

To my friends and colleagues at work for their encouragement, support and advice during this journey.

ACKNOWLEDGEMENTS

I am eternally grateful to God almighty for seeing me through this long and tough journey. This far it has been the Lord’s grace that has enabled me to get to this point. Completion of this research project would not have been possible without Gods help.

I also take this time to express my sincerest gratitude to my research project supervisor, Dr. Elisha Abade for his invaluable advice and corrections throughout this research. His guidance and feedback have been invaluable throughout this process.

Contents

1 CHAPTER 1: INTRODUCTION 10

1.1 BACKGROUND 10

1.2 PROBLEM DEFINITION 12

1.3 RESEARCH OBJECTIVES 12

1.4 RESEARCH QUESTIONS 13

2 CHAPTER 2: LITERATURE REVIEW 14

2.1 SECURITY PROBLEMS IN SDN 15

2.2 DOS AND DDOS ATTACKS – DEFINITION 16

2.3 DDOS TYPES 17

2.3.1 Volume-based DDoS 17

2.3.2 DDoS using Protocols 17

2.3.3 DDoS using Applications 17

2.4 Other DDoS Attack Types 18

2.4.1 User Datagram Protocol Flooding 18

2.4.2 SYN Flooding using TCP 18

2.4.3 Ping Flooding 19

2.4.4 Smurf Flooding 19

2.5 Distributed Denial of Service Detection and Mitigation 19

2.6 Classifying DDoS Defense Mechanisms 20

2.6.1 DDoS Attack Prevention 20

2.6.2 Distributed DOS Attack Detection 20

2.6.3 DDoS Source of Attack Identification 21

2.6.4 Distributed Denial of Service Attack Reaction 21

2.7 Firewalls 21

2.7.1 Firewall Types 21

2.8 Firewalls against Denial of Service and Distributed DoS 22

2.9 Software Defined Networking vs Firewalls 23

2.10 SDN AND DDOS ATTACKS 23

2.11 RELATED WORK 24

2.12 Conceptual Architecture 29

2.13 Tools for Research 32

2.13.1 Controllers 32

2.13.2 Network Emulator 33
2.13.3 Hypervisor 34
2.13.4 Open vSwitch 34
2.13.5 Packet Generation 35
2.13.6 sFlow-RT 35
2.13.7 OpenFlow 36
3 CHAPTER 3 METHODOLOGY 37
3.1 Introduction 37
3.2 Prototype Design 37
3.2.1 Experiment Requirements 37
3.2.2 Network Setup 38
3.3 Application Development 39
3.3.1 Software Development Model 39
3.3.2 Requirements Definition and Analysis 39
3.3.3 Application design and development 39
3.3.4 System Testing 40
3.3.5 Implementation 40
3.4 Experimentation 41
3.4.1 Fixed parameters 41
3.4.2 Variable parameters 41
3.5 Research Tools 41
3.5.1 Controller 41
3.5.2 Network Emulator 42
3.5.3 Open vSwitch 42
3.5.4 Packet Generation 43
3.5.5 Hypervisor 43
3.5.6 sFlow-RT 43
3.5.7 OpenFlow 44
3.6 Research Schedule 45
3.7 Experimenting with the setup 45
4 CHAPTER 4 RESULTS AND DISCUSSION 48
4.1 Experiment Results 48
4.2 Starting Floodlight 48

4.3 Starting sFlow 48
4.4 Starting Mininet 48
4.5 Configuring h1 as a web server 52
4.6 Simulating traffic on the network 54
4.7 Ping Flood Simulation: 55
4.7.1 Normal traffic: 55
4.7.2 CPU Usage on Server under normal traffic conditions: 58
4.7.3 With no Ping Flood Mitigation: 60
4.7.4 CPU Usage on Server: 61
4.7.5 Running the mitigation: 64
4.8 TCP SYN MITIGATION: 69
4.8.1 Syn attack with nping: 73
4.8.2 CPU usage on server: 74
4.8.3 TCP Connections on Server: 75
4.8.4 Graph on network: 76
4.8.5 Web server Access: 77
4.8.6 SYN Mitigation 77
4.8.7 Usage on network 79
4.8.8 SYN Attack with hping3 81
4.8.9 Total server failure: 85
5 CHAPTER 5: CONCLUSION 87
5.1 Future Work 91
6 CHAPTER 6: REFERENCES 92

Figure 1 A High Level Schema of the SDN Network 14
Figure 2 Statistical method for DDoS prevention 25
Figure 3 DDoS prevention framework 27
Figure 4 Conceptual Architecture 30
Figure 5 Conceptual Architecture 31
Figure 6 OpenvSwitch 35
Figure 7 Simulation Network Topology 38
Figure 8 OpenvSwitch 42
Figure 9 sFlow-RT Operation........ 44
Figure 10 Project Schedule 45
Figure 11 Simulation Network Topology 46
Figure 12 Starting Floodlight Controller........ 48
Figure 13 Starting sFlow........ 48
Figure 14 Starting mininet 52
Figure 15 Accessing webserver 54
Figure 16 Normal traffic........ 55
Figure 17Normal topology traffic........ 56
Figure 18 Normal protocol traffic 57
Figure 19 Normal traffic........ 57
Figure 20 CPU usage on server 58
Figure 21 htop CPU output 58
Figure 22 Tasks, threads 59
Figure 23 htop detailed process information. 59
Figure 24 Ping flood attack launched........ 60
Figure 25 CPU usage on server 61
Figure 26 Traffic on network........ 62
Figure 27 Protocol traffic during ping flood attack........ 63
Figure 28 Traffic on network........ 63
Figure 29 Ping flood mitigation script........ 64
Figure 30 Traffic drop after ping flood mitigation 64
Figure 31 Topology traffic after mitigation........ 65
Figure 32 Total traffic after ping flood mitigation 66
Figure 33 Protocol traffic after mitigation 67
Figure 34 CPU usage after mitigation 68
Figure 35 CPU usage on server before attack........ 69
Figure 36 Traffic on network before tcp syn attack........ 69
Figure 37 Topology traffic before TCP SYN attack 70
Figure 38 Network traffic........ 71
Figure 39 Total protocol traffic 71
Figure 40 Accessing web page 72

Figure 41 TCP connections on server........73
Figure 42 TCP SYN attack with nping........73
Figure 43 CPU usage on server........74
Figure 44 TCP connections on server........75
Figure 45 Graph showing an attack on the server........76
Figure 46 Web access during attack........77
Figure 47 Mitigation of attack........77
Figure 48 Connections on server........78
Figure 49 Traffic drop after mitigation........79
Figure 50 CPU usage........80
Figure 51 TCP SYN attack with hping3........81
Figure 52 server connections........82
Figure 53 CPU usage on server........83
Figure 54 Web server access........84
Figure 55 Ping to server........85
Figure 56 Server failure........86
Figure 57TCP SYN attack mitigation........91

ABBREVIATIONS

RAM - Random Access Memory

IP Internet Protocol

DoS Denial of Service

ARP - Address Resolution Protocol

DNS - Domain Name System

ONF - Open Networking Foundation

REST - Representational state transfer

ICMP Internet Control Message Protocol

SOM - Self-Organizing Maps

SDN Software Defined Networking

ACL – Access Control List

API - Application Programming Interface

SYN synchronization

OSI – Open System Interconnection

UDP User Data Protocol

HTTP - HyperText Transfer Protocol

ONOS – Open Network Operating System

TCP Transmission Control Protocol

DDoS Distributed Denial of Service

1 CHAPTER 1: INTRODUCTION

1.1 BACKGROUND

A Software Defined Network is one in which switches do not process incoming packets (Open Networking Foundation (ONF), 2017). SDN's main characteristic is the separation of the network into three layers. Layer 1 is the infrastructure layer constituting unintelligent network equipment such as switches. These simply drop and/or forward packets depending on the flow information installed onto them by the next layer – the control layer. The control layer links applications running on a layer above to the lower layer which is the infrastructure layer. The final and top layer is the application layer; it's composed of applications that communicate with lower layers through an application programming interface.

Incoming packets are assessed by devices in the infrastructure layer. They always look for a match and the corresponding action e.g. forward, drop etc. If no match is found, the packet will then be sent to the controller for further processing.

During denial of service attacks (DOS), attackers will try to prevent users of a system or service from obtaining it normally. When the attackers use more than one source or machine to carry out their malicious activities in a coordinated manner; then it is termed a distributed denial of service (DDoS).

We can classify DDoS attacks into three broad categories (Kesavan, 2016):

1. Volume-based distributed denial of service attacks.
2. Protocol-based distributed denial of service attacks.
3. Application-based distributed denial of service attacks.

Volume-based ddos attacks entail using massive amounts of traffic resulting in saturation of the bandwidth of the host that has been targeted.

Protocol attacks take advantage of vulnerabilities in the internet and transport layers of the OSI model thus making the targeted host inaccessible.

DDoS attacks exploiting weaknesses in the OSI application are known as application attacks. These attacks are very sophisticated and also the most challenging to identify and mitigate.

DDoS defense mechanisms can be classified as:

1. DDoS Attack Prevention – this mechanism aims to stop any attack before it compromises the target.
2. Distributed DOS Attack Detection – the main aim is to ensure the attack is detected when it is in progress.
3. DDoS Source of Attack Identification – the source of the attack is identified.
4. Distributed Denial of Service Attack Reaction – repercussions of the attack are eliminated as soon as possible.

Firewalls are commonly deployed to protect networks from attacks such as denial of service attack and distributed denial of service attacks. Firewalls are network devices or systems that control the flow of traffic between networks. There are two major types of firewalls that are used currently:

a. Network Layer firewalls – these firewalls use protocol/port numbers, source/destination IP addresses and other parameters filter packets based on protocol, source and to filter packets.
b. Application Layer firewalls – these firewalls use the OSI application layer for their operation.

Software defined networking (SDN) features can greatly improve firewalls operation as described below:

- SDN enables the target systems to tailor their defense according to their goals and weak points.

- SDN is similar to network firewalls in that policies can be set using the flow tables in the controller.

- A firewall will use access control lists whereas with SDN policy is defined with the flow rules which are pushed to the switches. SDN's flow tables offer more options for filtering compared to access control lists.

- SDN's centralized controller manages all the switches in a topology thus enhancing scalability unlike network firewalls which are setup for each device in a network.

There has been a lot of research focusing combatting DDoS attacks in Software Defined Networks. SDN has been used in conjunction with other technologies such as SOMs (self-organizing maps); support vector machines, entropy, statistical methods etc. to detect and mitigate DDoS attacks.
However, these methods seem to be generalized and mainly address the volumetric attacks. This research aims to build on previous work that already addresses volumetric attacks, to find a solution for protocol based attacks. Combating these attacks entails understanding the working of the various attacks and designing a specific solution geared towards detecting and mitigating the specific attack.

1.2 PROBLEM DEFINITION

Frequency and sophistication of distributed denial of service attacks (DDoS) is rapidly growing. These DDoS attacks are the most popular malicious attacks (Pescatore, 2014). Previous research work like (Mousavi, 2014), and (Luo, Wu, Li, & Pei, 2015) has focused mainly on volumetric DDoS attacks thus leaving protocol and application DDoS attacks unexplored. This research will build on previous research work on volumetric DDoS attacks to find a solution for protocol based DDoS attacks by taking advantage of the SDN architecture. The research will focus on TCP SYN flood attacks and ping flood attacks.

Software Defined Networking's centralized control offers a better way to mitigate these flooding attacks. Using SDN, we can have the application layer hosting a new set of security services that take advantage of the global view of the network from the controller and the capability to directly control the switches (Cabaj, Wytrębowicz, Kukliński, Radziszewski, & Truong , 2014).

1.3 RESEARCH OBJECTIVES

This research will study SDN aiming to find vulnerabilities with regards to DDoS attacks.
The main objectives are:

a) Establish methodologies and technologies that we can apply for detection and prevention of protocol based distributed denial of service attacks (DDoS).

b) Develop an application with the capability for detection and stopping protocol based distributed denial of service attacks.
c) Evaluate accuracy of the developed application in terms of detecting and deterring protocol based DDoS attacks.
d) Determine whether the application can handle malicious traffic while normal traffic is left unaffected.

1.4 RESEARCH QUESTIONS

This research aims to explore a possibility of combating DDoS attacks based on various protocols using Software Defined Networking.

Research questions are as follows:

1. What technologies and methodologies can we use fore detection and prevention of protocol based distributed denial of service attacks.
2. Can we code a system or application to detect and stop protocol based DDoS attacks?
3. Is the application developed able to detect and deter protocol based DDoS attacks?
4. Can the application developed detect and stop DDoS while allowing normal traffic to continue flowing unaffected?

2 CHAPTER 2: LITERATURE REVIEW

Software defined networking is a new networking technology that has recently emerged. It provides an architecture separating the control and the data planes. The central control plane has an advantage in that our network becomes much more dynamic thus making resource management efficient and cost effective.

SDN has the unique feature of abstracting a network into two planes - control and data. The depiction of an SDN is shown below. There are three very distinct layers:

Infrastructure Layer - lowest layer consisting of simple network devices such as switches. These do not have any intelligence. Their core function is either to drop or forward packets as dictated by the flow table rules on the control plane through a chosen southbound protocol.

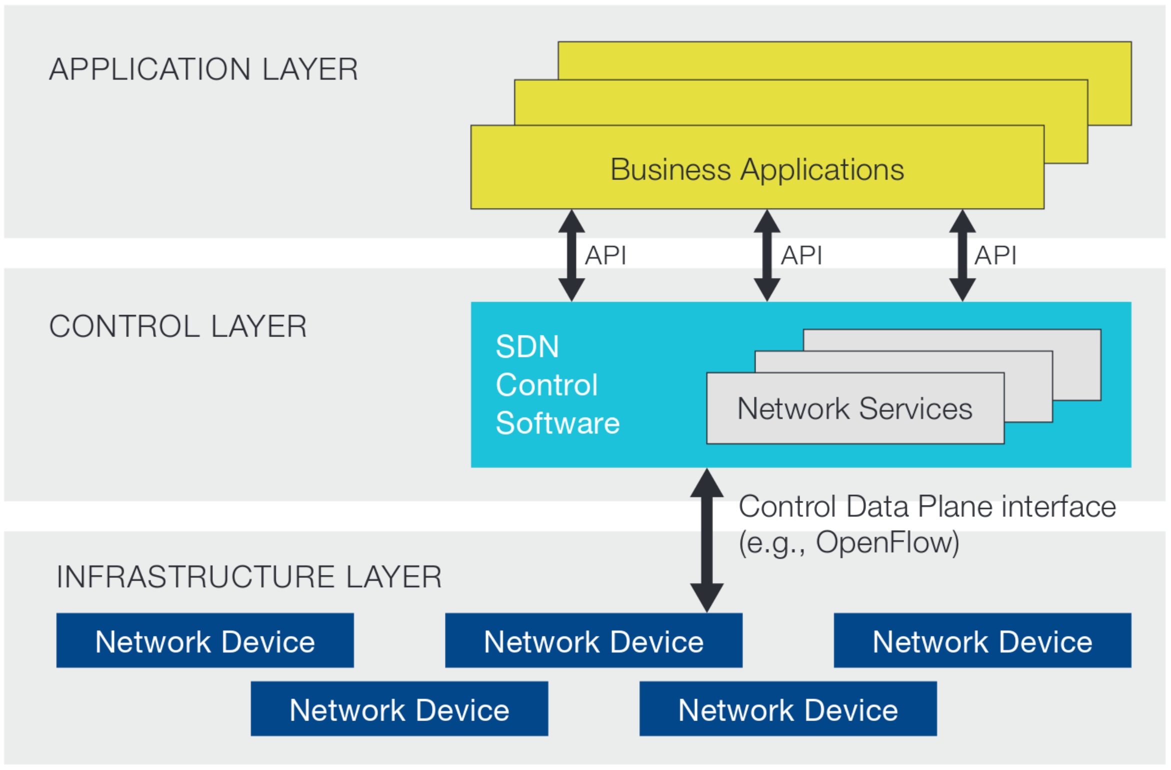


Figure 1 A High Level Schema of the SDN Network

Control Layer - this sits above the infrastructure layer. Its main function is to interface applications running on a higher layer with the dumb devices below in the infrastructure layer. It also has what is termed as the brain of the SDN - the controller. This layer has the privilege of having a centralized control and view of the SDN. APIs allow applications above to control devices below.

Application layer - has applications running. Some manage the network below and some secure the SDN. Applications use a chosen northbound API of the controller to talk to the controller. These applications take advantage of the central view of the SDN by the controller to understand the whole network. This data is then used to come up with flow tables to be pushed to the dumb switches.

SDN is continuously evolving while facing many challenges. Some companies like Google and Huawei have already deployed SDN.

SDN is extremely unique because it offers capability for programmability. This can be achieved due to the separation of the two planes - control and data. Devices on the SDN are dumb and programmable.

Since data and control planes are separated:

- The network control can be handled separately without affecting flows of data.
- Devices lack intelligence. This is thus put in the controller.
- Dumb devices are controlled from higher layer using software.
- The applications have a wide range of capability to define how the network could act or operate.

2.1 SECURITY PROBLEMS IN SDN

In SDN some of the main priorities are the reliability and security due to the cyber-attack and growing trend of digital threats. Commercial adaption of SDN is still in its early phase although Google, Microsoft, Yahoo have been conducted a lot of experiments and research on it. A main concern today is the accessibility of Internet routers thanks to the widespread of clouds in terms of reliability. If SDN control platforms want to become an essential part of network applications reaching a high level of availability is crucial. DDoS attacks are a major threat to this goal as their main aim is to make services unavailable by exhausting computing resources.

2.2 DOS AND DDOS ATTACKS – DEFINITION

Denial of Service attacks are usually originated by malicious individuals with the aim of interrupting regular or standard functioning/operation of a service or application. In case the malicious user incorporates many machines/hosts in the attack, then we refer to it as a distributed denial of service. Botnets - this is large pool of devices which are able to communicate, are usually used. Botnets are usually infected and the malicious users has remote control of them. An attack is achieved when the botnets each send traffic to a target resulting in exhaustion of critical resources thus making them unavailable to legitimate users. The attacker has the potential to send massive volume of packets to the target with spoofed source IP addresses. Later In this section we are explaining in details the nature of the botnet.

Distributed denial of services are the top security issue on the internet and web servers are the main target. As much as we understand what DDoS is, its detection is very complicated due to complexity of differentiating normal and malicious traffic.

Detecting DDoS based on flooding is a major challenge for security of the internet now. This attack relies heavily on large resource imbalance between the internet consisting on millions of devices and the limited resources on a victim host when handling unusually large number of requests that are not genuine resulting in normal user requests not being processed because resources are exhausted. (Rodrigo, Edjard, & Alexandre, Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow, 2010).

These attacks are feasible due to easily available tools that are easy to use e.g. (Dittrich, 1999). Users with little expertise in networking can use these tools to set up an attack to a targeted host. Use of IP spoofing makes it very hard to trace source of attack. Recently the twitter website fell victim to DDoS and had its services unreachable for a number of hours. (Sutter, 2009).

When trying to detect a DDoS there are a number of challenges. First is the fact that packet headers have been modified and made to look normal thus differentiating between traffic that is safe and legitimate and that from malicious hosts is extremely complicated and hard. Secondly, data to be analyzed is huge. These two issues make detection extensively complicated and response timelines get even worse.

2.3 DDOS TYPES

Distributed denial of services can be achieved in various ways. Classification of these attacks can be done using the traffic characteristics, amount of packets/data used, and what weakness was targeted on the host. Based on these, we can have three distinct types of DDoS i.e. volumetric, protocol based and finally application based attacks. (Kesavan, 2016)

2.3.1 Volume-based DDoS

These are by a long shot the most popular attacks accounting for close to 65% of DDoS incidents based on a report from Arbor Networks. Even though they usually have a large size of traffic that could even be in hundreds of Gbps, they do not require a similarly large traffic to be originated by the attacking hosts. This characteristic enables them to be very simple to generate. A magnifying medium could be used to convert little data traffic into gigabytes of end traffic on the targeted host. This method usually sends normal requests to a service such as DNS most times using a fake source address. When the service responds to this request, they are in effect communicating with a fake address resulting in the targeted host being overwhelmed with exaggerated traffic.

A true query to the DNS from a fake source IP for ANY records can multiply the traffic even 100 times as the request could respond with all records such as A, CNAME, NS, or MX. A similar request sent to multiple DNS serves e.g in the range of a hundred or a thousand can lead to generation of gigabits of packets; the targeted host will collapse. (Kesavan, 2016).

2.3.2 DDoS using Protocols

These mainly use vulnerabilities in network and transport layers. The most popular being TCP based SYN flooding which uses a successive TCP SYN packets sent to a host thus making it unable to operate normally henceforth. (Kesavan, 2016).

2.3.3 DDoS using Applications

This type of DDoS is the most elusive and most complicated to identify or even tackle. Malicious users in these attacks are experts having intricate knowledge of how the targeted application operates. Generated packets for attack are usually safe but target the OSI application layer; it triggers a process in the layers inner working which takes up all the resource resulting in unavailability. This is why it is nearly impossible to tackle this kind of attacks.

2.4 Other DDoS Attack Types

There are other different forms of DDoS attacks. These are introduced below.

2.4.1 User Datagram Protocol Flooding

User datagram flooding or UDP flooding is characterized by malicious hosts making the targeted host incapacitated by use of massive UDP packets sent at a very high rate such that they cannot be processed.

Since UDP can use random ports, targeted ports vary making prediction very difficult. Victim host receives the UDP packets, processes them to find associated process on the destination port. Failure to find the associated process, a destination unreachable ICMP message is generated directed to the attacking host. This whole process consumes massive resources on the host thus eventually making them unavailable once all resources are engaged.

2.4.2 SYN Flooding using TCP

TCP SYN flooding take advantage of the thre way handshake mechanism used in TCP. With TCP, this handshake process is used to establish a connection between two hosts before the start exchanging data.

The host initiating communication first sends a SYN request to the host it is interested in communicating with. Once the recipient gets the SYN packet, it opens a port for a new process and continues to listen for more packets. A SYN-ACK is sent back to the initiating host. This shows it's expecting more from the host.

On receiving the SYN-ACK, the initiator sends back an ACK thus completing the 3 way handshake and communication can then proceed.

TCP SYN flooding happens after an attacking host sends multiple SYN requests which the target handles normally by dedicating ports and other resources such as memory and CPU. The attacker never sends the SYN-ACK thus leaving the recipient expecting messages that won't come. This expectant state continues for some time until the configured clock timeout upon which the TCB is freed.

When many SYN requests are sent and the host is in a state where new TCBs cannot be allocated to genuine requests, the host is unresponsive. (Wesley M. Eddy, 2006).

2.4.3 Ping Flooding

In Ping flooding, an attacker sends numerous ICMP echo reqests which it cannot service. This attack targets bandwidth of the network instead of computer resources since a host can process ICMP very fast with minimal compute resources. (Gupta, Joshi, & Misra, 2009). Once the bandwidth is exhausted, the targeted host becomes unreachable hence unresponsive.

2.4.4 Smurf Flooding

Smurf flooding entails the target host e.g. server being flooded with echo responses/replies instead of echo requests. (Gupta, Joshi, & Misra, 2009). The attacker basically makes the source IP address of the initial echo request that of the target. A broadcast of the echo request is made to a number of devices. These then construct replies which are sent to the masqueraded IP address, which is the IP of the victim hence the attack takes place.

2.5 Distributed Denial of Service Detection and Mitigation

There are well established mechanisms for protection against DDoS and mitigation of DDoS attacks when they happen to occur. (Haris, Ahmad, & Ghani, 2010). They involve:

Prevention of DDoS

Detecting DDoS

Filter-based mechanisms

Determination of DDoS sources.

Firewalls based on destination addresses combined with filtering of packets has widely been used with exceptional success. (Hussain, Heidemann, & Papadopoulos, 2003). This involves placement of a device capable of monitoring at the section the device to be protected connects to our network. The whole network benefits from the protection. Unfortunately, this method has no capacity to learn and implement protection from the new knowledge. This shortcoming makes the solution untenable in cloud based solutions where protecting each destination individually is quite impractical.

Contrastingly, SDN enables global view of the whole network thus allowing complete network based detection.

Network based DDoS detection involves placing a number of monitors over the entire network. These monitoring agents have predefined signatures; the monitors constantly attempts to match traffic passing through the network against the signatures. This activity does not disrupt the traffic flow.

This detection method does not feature any learning but relies on analysis developed externally to come up with required signatures to be used by monitoring agents.

2.6 Classifying DDoS Defense Mechanisms

Defending DDoS is a major challenge for network administrators. A number of options exist to achieve this:

1. DDoS Attack Prevention – this mechanism aims to stop any attack before it compromises the target.
2. Distributed DOS Attack Detection – the main aim is to ensure the attack is detected when it is in progress.
3. DDoS Source of Attack Identification – the source of the attack is identified.
4. Distributed Denial of Service Attack Reaction – repercussions of the attack are eliminated as soon as possible.

2.6.1 DDoS Attack Prevention

Aims to stop the attack early in its process before any damage is experienced. Traffic recognized as malicious is denied based on known patterns. Below approaches can be used to stop DDoS:

Traffic filtering - examining and analyzing incoming and outgoing traffic on all routers.

Firewall - performs traffic filtering based on unique characteristics such as protocols, source and destination addresses, source and destination ports etc.

2.6.2 Distributed DOS Attack Detection

Mechanisms used in this category should have the capability to determine whether traffic is actually legitimate or malicious. The mechanism should be accurate to avoid legitimate traffic being confused for malicious traffic; this is known as false positives. (Carl, Kesidis, Brooks, & Rai, 2006).

Efficient mechanisms will eventually have a good balance between false positives and false negatives. We have mainly two schemes: (Mirkovic, Martin, & Reiher, 2010), (Srivastava, Gupta, Tyagi, Sharma, & Mishra, 2011):
Pattern detection - attacks are identified through comparison of traffic coming in with well-defined attack signatures that are stored in a filesystem or storage.
If a successful match of an attack to a signature is made, the method could be very accurate with very few false positives. Issues will arise in case new forms of attack are generated or slight variations that can trick the mechanism.
Anomaly detection - this will identify an attack by detecting anomalous traffic patterns.
Rate-based detection - when flow of traffic in a network changes, this method will detect any issues by using known patterns of traffic volumes based on time.

2.6.3 DDoS Source of Attack Identification

When the system identifies an ongoing attack, the best course of action would be to block that associated traffic at its source.

2.6.4 Distributed Denial of Service Attack Reaction

This mechanism aims to stop or minimize effects of a DDoS and eliminate the malicious traffic while leaving the legitimate traffic unaffected. This minimizes damage caused by DDoS while it is in progress.

2.7 Firewalls

Firewalls are network security systems used to control the flow of traffic that is inbound or outbound between networks. They analyze traffic by looking at characteristics such as layer 2/3 headers - hardware address, source/destination addresses.

2.7.1 Firewall Types

Two broad categories of firewalls exist; this is mainly based on the layer of the network at which it operates. These are - network layer firewalls and application layer firewalls.

2.7.1.1 Network Layer firewalls

This type of firewall is also known as packet filtering firewall. Packets are processed and analyzed based on ports or protocol in use, MAC address, layer three source and destination address. This mechanism is usually deployed on switches and routers. ACLs are mostly used for this task.

An ACL will permit or drop packets using specific rules that have been configured and applied to ports or addresses available on the host. ACLS work on traffic in both directions.

One major drawback of ACLs is that they are not adaptable to conditions. Once configured, they do not adapt to changing conditions. Secondly, some attacks could use well known port numbers and could then pass via the firewall undetected.

2.7.1.2 Application Based firewalls

These types of firewalls operate at the top layer of the network i.e. application layer. However, they are slower than packet filtering firewalls as they have to maintain state information for applications at both the client and server. They also continue inspecting traffic.

2.8 Firewalls against Denial of Service and Distributed DoS

Firewalls are second in the most targeted systems on the internet after common servers when it comes to DoS and DDoS. At time access control lists could contain thousands of lines to match traffic for the purpose of filtration between legal and non legal traffic. The main disadvantage with this is that DDoS attacks nowadays will be camouflaged in normal traffic and when firewalls allow connections using its configured access control lists for each sent packet. This finally results in connection tables being exhausted. When this limit is reached, new connection requests will be dropped irrespective of their legitimacy.

This phenomenon is a consequense of firewalls not being designed for distributed denial of service. Firewall devices process packets one at a time against the configured access control lists hence they do not the capability to differentiate malicious/suspect traffic and the normal or legal traffic thus resulting in a weak point.

2.9 Software Defined Networking vs Firewalls

Current firewall systems have many shortcomings when it comes to defending denial of service attacks since they focus on allowing or denying traffic using configured access control lists. This makes it complicated to determine which traffic is to be allowed and which is to be dropped. SDN technology in conjunction with OpenFlow can improve on firewall functionality. Features to assist us are:

-Since the control and data planes are separated, the network devices are abstracted allowing software engineers to create applications using high level programming languages to manage the network infrastructure.

SDN has enabled capability to develop of unique defense strategies that depend on the target's unique goal and its weaknesses.

This is accomplished using entries in a table with header or packet details to filter incoming/outgoing traffic.

Whereas a firewall will use access control lists, SDN's OpenFlow devices deploy flow tables. Access control lists are handicapped by number of lines that can be used to match and filter traffic. Typically ACLs will match using source/destination IP addresses, port then define at action. SDN's OpenFlow tables make use of more fields with regards to protocols and Open System Interconnection layers.

Hence, with OF flow tables, we are able to come up with broader and also clearly drilled down characteristics of the traffic.

With Software Defined Networking, switches are centrally managed by the controller. This gives SDN an advantage over network based firewalls which have each device with its own control plane by allowing administration of traffic and flow policy at a single point. This allows for scalability. This characteristic is also makes SDN better at DoS/DDoS as traffic is followed as a whole.

2.10 SDN AND DDOS ATTACKS

SDN uses the central controller to its advantage to make decisions concerning the whole network using flow tables.

2.11 RELATED WORK

(Mousavi, 2014), used entropy to detect DDoS attacks. Entropy (Shannon-Wiener index) is defined as the measurement of randomness with regards to a randomly varying data and for this research, traffic traversing over the network using destination internet address.

His method sought to identify the attacks in their early stages thus mitigating them in good time. The controller normally collects statistics from all switches so that it can find flows not in use.

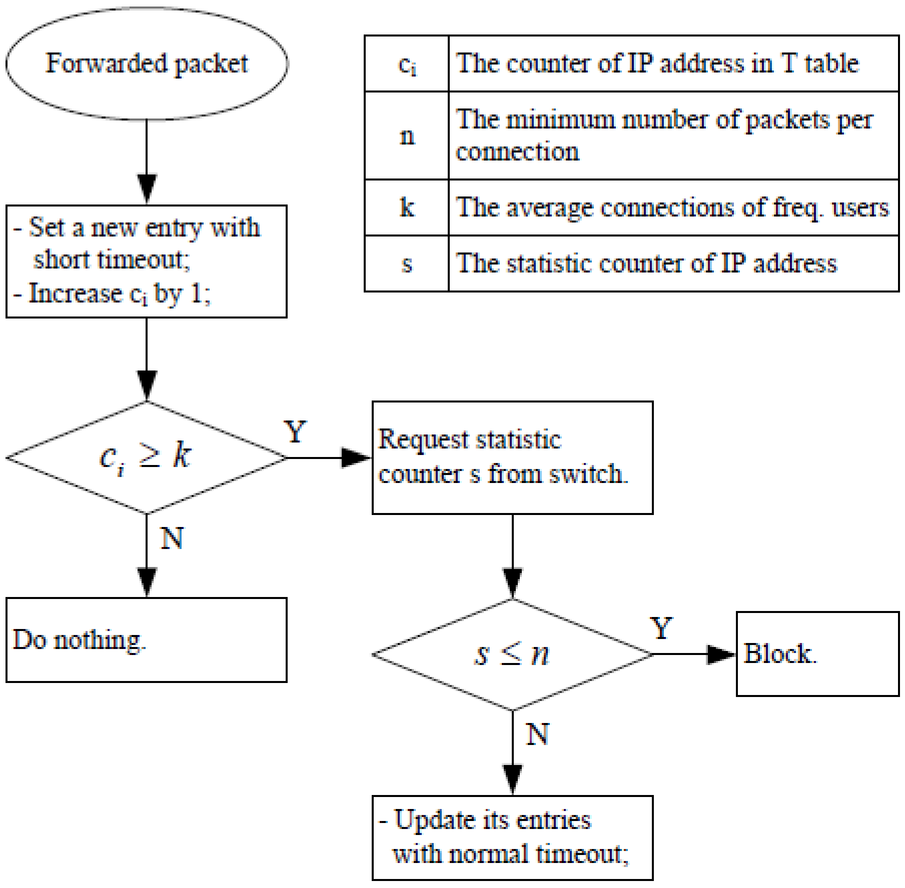
They are usually deleted if not traffic/packets are received over a given time, known as time out and can be set do different values.

Mousavi proposed adding additional statistics onto the SDN's controller. These were - destination address entropy on the SDN controller. The module will detect if the rate of packets coming in to a given destination is deemed very high compared to the normal rate.

50 was set as the size of the window i.e. every 50 of Packet in packets are analyzed to check the destination address then entropy is calculated. The result is compared to the threshold that was set. An entropy that is below the threshold and continues for 5 consequent periods, then it is concluded that a DoS attack is happening. If this happens with five entropy bands, which is two hundred and fifty packets of a flow thus it gives the network ability to detect an attack eraly enough.
The above method assumes traffic is evenly distributed to all IP addresses in the network. This would not apply in a server based environment. There could be more traffic going to a server e.g. a web server. Setting the window size and threshold is not straight forward as all networks are not the same. Networks are also not static hence the values might need to be dynamic. My research overcomes the above challenges by designing a solution that protects a particular server and can be applied to any service that an administrator would like to protect. The solution works in real time hence does not need setting values such as window size or thresholds.

(Dao, Park, Park, & Cho, 2015) Uses a statistical method based on the fact that majority of users send a minimum of five packets to any destination at a time. Users considered suspicious will send less than five packets at a time.

n denotes the smallest number of packets sent by a frequent user for each connection. The average of setup connections will marked by k.


Forwarded packet
- Set a new entry with short timeout;
- Increase c_i by 1;
$c_i \geq k$
Y
N
Request statistic counter s from switch.
Do nothing.
$s \leq n$
Y
N
Block.
- Update its entries with normal timeout;

c_i	The counter of IP address in T table
n	The minimum number of packets per connection
k	The average connections of freq. users
s	The statistic counter of IP address

Figure 2 Statistical method for DDoS prevention

T is used to represent a table construct on the SDN controller. It's used to keep the layer 3 address of each packet that traverses the switch. For each layer 3 address, a counter, ci, is used to keep a track of arriving packets.

If an attack was in progress, the SDN controller will assume that there is a DoS/DDoS attack ongoing as soon as a packet is forwarded by a switch and seen by the controller.

SDN controller will create a new entrant with hard and idle timeouts; their values are not bigger than the values assigned for normal entries so that the lifetime is limited. The layer 3 source address in the table T is made up to date and the associated counter (ci) is increased by a single digit.

After ci reaches k, its data traffic characteristic is analyzed by obtaining s i.e. average packet numbers. When s>n, we conclude that the layer 3 source IP successfully set up a connection and

sent authentic data, meaning we can classify it as a frequent user. The SDN controller then sends a message to reset hard and idle timeouts of the current entries to standard values.
If s<n, the system concludes that this is unsafe traffic. The SDN controller will send a command to drop associated traffic.

This solution is based suited to volume based DDoS attacks. The solution would have challenges detecting subtle attacks based on protocol or application weaknesses. My research improves this statistical method by adding a way of detecting protocol and application attacks and mitigating the same based on the characteristics of the attacks.

In (Phan, Bao, & Park, 2016) traffic characteristics of flows to a server in an ISP core network in the event of normal traffic flow and during a DoS/DDoS attack are analyzed. A combined usage of support vector machines (J. & Cristianini, 2000) and Self Organizing Map (Kohonen, 1997) enhances the efficiency of classifying flows for the traffic running on the network. For this setup, SVM is deployed as a tool for supervised learning model that is capable of analyzing given data and notice patterns. Self-organizing map is known as an efficient algorithm for the purpose of un-supervised learning in artificial neural networks. This collaboration of SVM and SOM takes advantage of each algorithm used; not just for generation of accurate decisions but to ultimately bring down time spent processing. Additionally, SVM and SOM collaboration is applied in a proposal to limit distributed denial of service attack consequences.
After analyzing, a prevention mechanism is proposed based on SDN. This architecture can capture and analyze flows in real time. This is the input into the prevention step. As a last step, a prevention scheme based on multiple criteria was designed. This uses hard decisions thresholds and system of fuzzy inference with its main purpose being detection of a DDoS attack. If attacks are determined to be present, the suspicious flows can be dropped by this system.
They proposed a novel network architecture designed to prevent DDoS attacks. This architecture uses statistical characteristics of network traffic that is incoming.
SDN switches monitor below mentioned statistical characteristics then sends them to the controller.

- distribution of time between arrivals i.e. inter arrival time
- quantities of packets per flow distribution

- the sum of flow entries towards a server

An application running on the SDN controller uses the two algorithms i.e. hard thresholds and fuzzy logical scheme; is a response to the analyzation of statistical characteristics and determines if a DDoS attack progressing or normal state prevails.

However, for this method, the support vector machines and the self-organizing maps still have to be trained with datasets in order to be able to detect attacks. This can take time and could not be ideal for all situations. My method will use real time detection based on characteristics of the attacks hence will not need extensive training before it can work.

(Luo, Wu, Li, & Pei, 2015) Used the below framework:

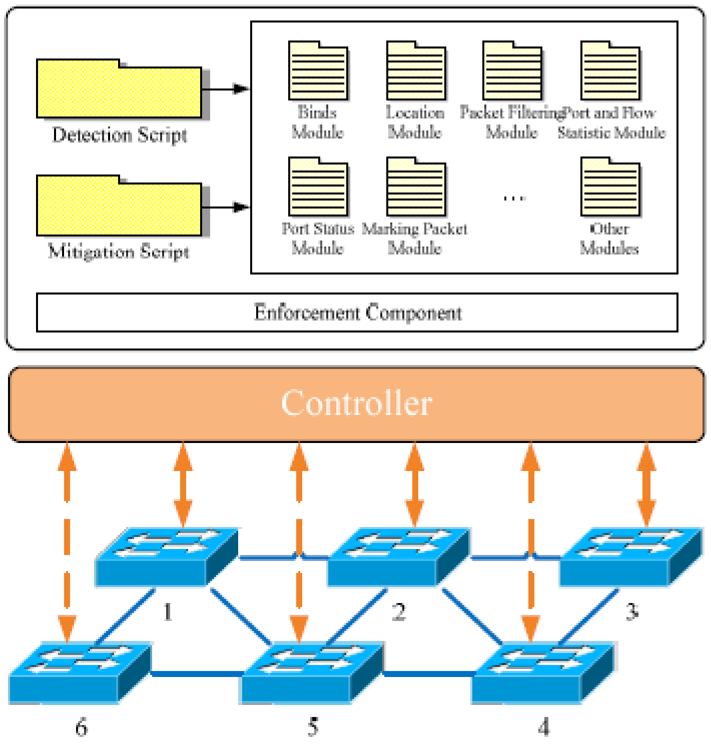


Figure 3 DDoS prevention framework

The framework has the following components:

- A scripting language.
- Modules that are already defined.
- A component to enforce.

Script language defines how to detect and mitigate. We have a number of predefined modules that can be used to come up with new methods. The framework has numerous conceptual and rudimentary modules. These are building blocks for detecting and mitigating DDoS.
The scripts are converted into corresponding flow rules for the SDN switches. These are installed into the switches using the system's enforcement module.
This enforcement module has a number of critical characteristics that make sure switch ruled obtained out of security policies are made a priority and installed/executed before other flows derived from other applications.
Below are responsibilities of the enforcement module:

- Rule source identification
- Detecting rules that are conflicting
- Resolving the detected conflicts

Some key modules of the framework are explained below:

1. Binding module - Any node could initiate a threat in our network thus making identification of a particular node in the network critical. Taking advantage of the SDN controller's centralized control and whole view, this module collects data from all equipment in the system e.g. IP address. This enables easy monitoring of the network.
2. Location tracking module - Knowledge of the each host's location eases blocking of detected attacks. SDN controller's centralization and global management enables this.
3. Packets filtering module –
4. Interface/port statistics querying function - for detection of DDoS attack, the system should have the capability to identify happenings associated with distributed denial of service attack incidents. Port data for each SDN switch is collected at intervals to determine both byte and packet rates. These values are used to make a decision.
5. Flow statistics queries module - This framework checks at intervals the SDN switches' flow entries to determine presence of flow entries considered malicious, removes the flows and installs new ones while blocking the DDoS.

The solution had many modules which have a toll on the processing capacity of the controller. The solution is also not designed for specific protocol and application DDoS attacks which are often subtle and difficult to detect. This research is an improvement on this solution as it aims to find a solution for these subtle protocol and application attacks.

(Rodrigo, Edjard, & Alexandre, Light-weight Distributed DoS Flood Attack Detection Using NOX & OpenFlow, 2010) Used SOM for distributed denial of service detection. Three modules are used in the solution.
1. Flow collector - periodically collects information of flow tables installed on switches. These communications are done over a secure channel.
2. Feature extractor - picks flows obtained from above. Isolates and processes characteristics pertinent to distributed denial of service detection; stores them in 6 tuples to be sent to the classifier.
3. Classifier - used to analyze 6 tuple to determine if it matches distributed denial of service attack or normal traffic. The classification could be achieved using either statistical or learning methods. For this research, SOM was used.

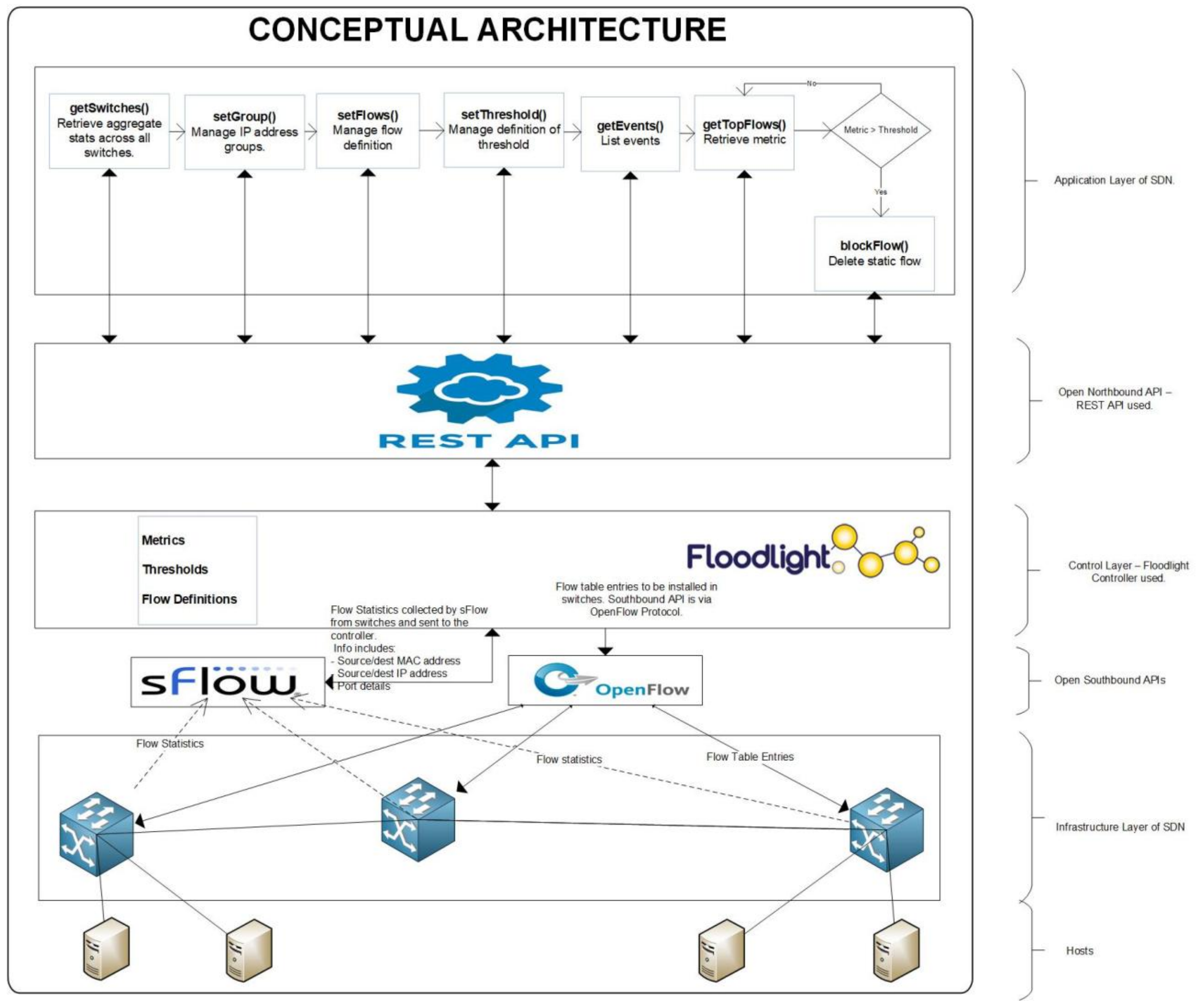
The Self Organizing Maps are trained using data collected from the SDN Open flow devices. Specific information used for the training include:

- average number of packets for each flow
- average traffic in bytes for each flow
- average time duration for each flow
- the percent of flow pairs
- the growth of a flow
- growth of a port

With time, the SOM gets better by improving stats of the vectors. This method also has the drawback of requiring extensive training of the SOM. This takes time and in the meantime, attacks can succeed while the training is still ongoing. This research's proposed method does not require any training.

2.12 Conceptual Architecture

Below is a simple conceptual framework of our solution:



CONCEPTUAL ARCHITECTURE
getSwitches() Retrieve aggregate stats across all switches.
setGroup() Manage IP address groups.
setFlows() Manage flow definition
setThreshold() Manage definition of threshold
getEvents() List events
getTopFlows() Retrieve metric
Metric > Threshold
No
Yes
blockFlow() Delete static flow
REST API
Metrics
Thresholds
Flow Definitions
Floodlight
Flow table entries to be installed in switches. Southbound API is via OpenFlow Protocol.
Flow Statistics collected by sFlow from switches and sent to the controller.
Info includes:
- Source/dest MAC address
- Source/dest IP address
- Port details
sFlow
OpenFlow
Flow Statistics
Flow statistics
Flow Table Entries
Application Layer of SDN.
Open Northbound API – REST API used.
Control Layer – Floodlight Controller used.
Open Southbound APIs
Infrastructure Layer of SDN
Hosts

Figure 4 Conceptual Architecture

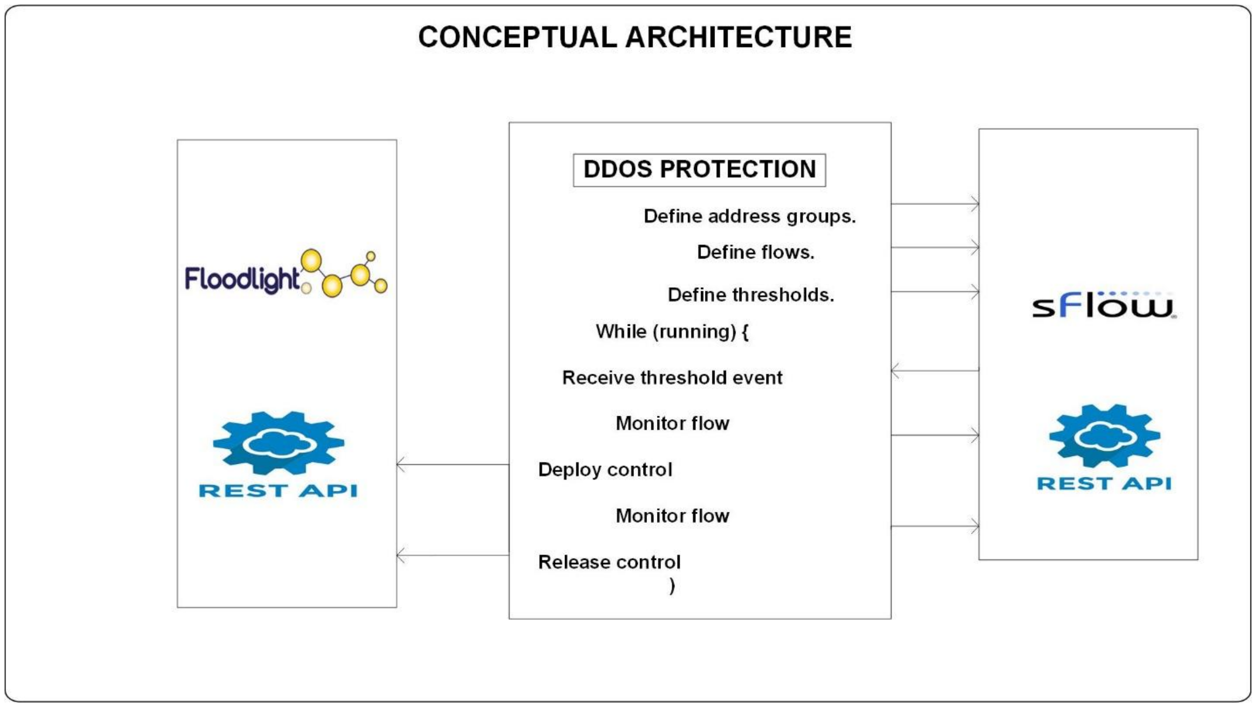


Figure 5 Conceptual Architecture

The conceptual architecture above depicts how the system will defend against Ping Floods and TCP SYN attacks. The system defends when there is a malicious flow by deleting the flow.

1. Definition of address groups – traffic is categorized into internal and external. Internal traffic is represented as an address block and is the address space used on the hosts and switches of the SDN. In our case it's the address block 10.0.0.0/8. External traffic is any other traffic that might be accessing our network. We define the whole address space as external traffic i.e. 0.0.0.0/0.
2. Flow definition - We define the flows by assigning names to packet characteristics identified and implemented in grouping the same packets into flows or keys, a unique value to associate the flow with and finally a filter for selection of specific traffic.
3. Defining thresholds - We define thresholds of packets that could signal a breach e.g. 10000 packets per second for a given flow.

4. Get threshold events - The application keeps polling for events in order to obtain notifications in case of new events.
5. Monitoring of flows - The flow is monitored and more information regarding the flow is obtained. This information is extracted from the event plus the agent, metrics and data sources.
6. Deploying control – after monitoring and determining that thresholds have been exceeded, the controller installs flows that result in traffic from the identified source being dropped.
7. Monitoring of flows – the flow is continuously monitored to ensure that the control measures took effect on traffic.
8. Releasing control - After a given time, the control action is stopped so that flow table entries engaged in blocking the attacker are released. If the attack happens again, another event will be generated and new control implemented.

To achieve the above, a node.js application will be created. Node.js was chosen because it allows for asynchronous input/output thus supporting higher levels of parallelism thus enabling SDN applications interact with many controllers, monitor systems etc. without blocking hence we get a quick and consistent response time.
sFlow and OpenFlow are used to create the test bed for the application to detect DDoS. Mininet allows us to create virtual networks.

2.13 Tools for Research

This section gives more information about various tools that can be used for the research.

2.13.1 Controllers

There are a number of controllers in the market today. These include:

ONOS Controller. Open Network Operating System - this is a project that is sponsored by The Linux Foundation and aims to come with an SDN OS for communication service providers. It is designed with scalability and high performance/scalability. (ONOS, 2018)

Open-Daylight controller. Open-Daylight controller - this project is a collaboration open source project also by The Linux foundation and aims to promote SDN and network function virtualization aka NFV. Java programming language has been used for this project. (OpenDaylight, 2018)

NOX controller - the oldest controller. Operates as network control platform providing an abstracted high level programmable interface for management purposes and also offers ability to develop applications to control the network.
NOX was initiated by Nicira networks which was eventually bought by VMware. (NOX, 2018)

Floodlight Controller developed by Big Switch Networks. Floodlight operates together with OpenFlow; managing network traffic in SDNs. (Floodlight, 2018)

For our research we will use Floodlight because it can handle larger flows. Floodlight is compatible with Linux, Mac OS and windows, and it has topology discovery. There are other controllers.

2.13.2 Network Emulator

We have various network emulators in the market today:

ns-3 - this educational/research network simulation for network systems. It's free for use. (ns-3, 2018)

netem provides network/system emulation so as to get ability to test protocols. Variables that can be tested - losses, delays, duplications and reordering. (netem, 2018)

The research will use Mininet (Mininet, 2017) as the network emulator. This is the common emulation system for SDNs. It can simulate a required network even on a simple laptop. It makes use of kernel namespaces. Kernel namespace can avail independent processes with standalone network interface, route table etc.
Mininet uses process based virtualizations to simulate SDN switch and host on a given kernel.

2.13.3 Hypervisor

Virtualbox hypervisor is available for free. It is owned by Oracle. This hypervisor may be installed on any operating system. Virtualbox will support all OSs on the guest virtual machine. (Virtualbox, 2018)

Linux KVM - Linux based. It allows for conversion of the linux kernel into a hypervisor thus enabling virtual machines to have access to the hardware directly.

VMware - This hypervisor is can be expensive to use due to high licensing costs. It offers open versions of VMware ESXi but they are limited with no advanced features as the paid versions.

Citrix's Xen is another virtualization tool with roots in the open source world. It is mainly free. However, for advanced features, you have to pay. Several commercial editions exist that give more advanced features for management purposes, automating and uptime/availability.

Microsoft Hyper-V was introduced by Microsoft. It is free for evaluation purposes only. It is not advanced as VMware in terms of number of products available.

For this research, Oracle Virtualbox will be used as it is open source and offers full features on the free version.

2.13.4 Open vSwitch

OpenvSwitch (OVS) performs multilayer switching and can be programmatically controlled by SDN controllers, this switching platform will be used in this study because it works will with both the Floodlight controller and Mininet.
OVS architecture is shown below:

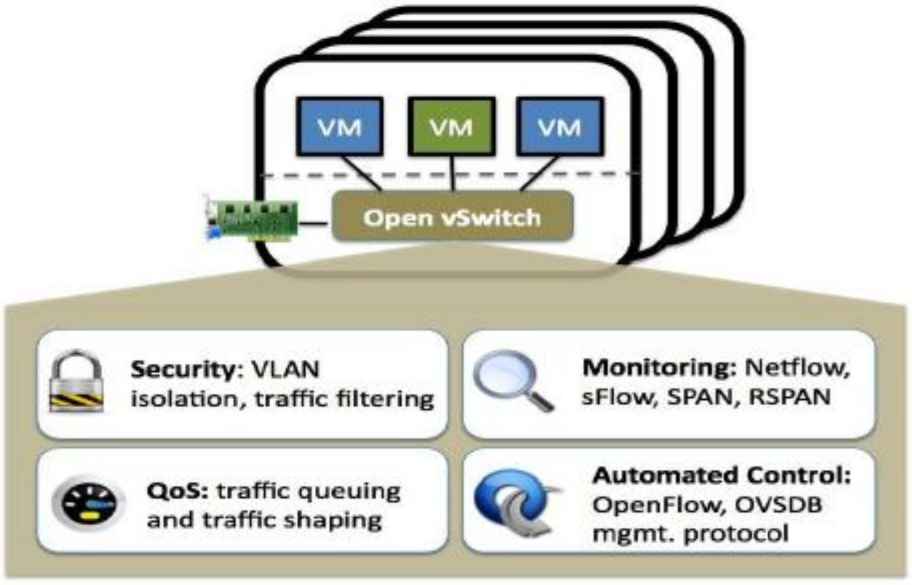

VM
VM
VM
Open vSwitch
Security: VLAN isolation, traffic filtering
Monitoring: Netflow, sFlow, SPAN, RSPAN
QoS: traffic queuing and traffic shaping
Automated Control: OpenFlow, OVSDB mgmt. protocol

Figure 6 OpenvSwitch

2.13.5 Packet Generation

Packet generation is done hping3 and nping. Hping3 has no graphical interface and is used as an OSI TCP-IP packet assembly and analyzation. In addition to ICMP pings, this tool can work on a variety of protocols such as TCP, User datagram protocol, raw internet protocol packets. Moreover, it contains the tracert mode and a unique ability to transport files over a secure channel etc.

On the other hand, NPING is a free tool used to generate packets, analyze responses, and measure response times. This tool is able to generate packets based on quite a big variety of protocols thus enabling users to have admin control of protocol headers.

Nping has a variety of functions e.g. simple ping, packet generation, stress testing, simulation of attacks such as DoS, traceroutes etc.

2.13.6 sFlow-RT

This project uses sFlow-RT to obtain real time visibility in terms of traffic and statistics, of the Software Defined Network. sFlow enables development of applications that are aware of the network's performance at any one time. This allows us to create applications with capability for DDoS detection and prevention, load balancing etc.

sFlow-RT has an analytics engine that obtains telemetry streams continuously from agents known as sFlow agents. These agents are found in network equipment, hosts and also apps. The agents convert this telemetry streams into metrics that can be acted upon and accessed via the REST API. This REST API enables administrators to define measurements, obtain given metrics, define thresholds, and get notifications. Applications can then be built to use this information as long as the language used supports HTTP/REST calls.
sFlow-RT enables administrators or users to have a view of applications, servers and network performance. This is achieved through a combination of network, host and app monitoring within an integrated analytics pipeline. (inMon, 2018).

2.13.7 OpenFlow

OpenFlow(OF) is a pioneer SDN standard that defines communication protocol enabling the controller to interact with the data plane of devices in the infrastructure layer of the SDN. SDN controllers can send configuration changes down to switches and routers using OpenFlow. Flow tables of the switches are modified by the controller using OpenFlow.

3 CHAPTER 3 METHODOLOGY

3.1 Introduction

In this chapter, a description of the methodology that was used in carrying out the study is discussed in detail. Tools and process followed are described.

Research methodology to be used is experimentation. Reasons for choosing experimentation:

1. Experiments are performed and the results we get indicate how effective the proposed distributed denial of service attack detection solution is.
2. Simulation of various experiments helps to obtain results as close as possible to a real world scenario. Final results can indicate whether the proposed method is adequate and effective for DDoS detection and prevention.
3. Since SDN provides a programmable network platform, experimentation can take advantage of this function. Experiments on a real environment can be achieved due to SDN's high configurability as it provides separation among virtual networks. Transitioning from experimental environment to an operational one can be done smoothly.
4. SDN separates the data and the control planes hence making it feasible to build big experiments on attacks and defenses.

3.2 Prototype Design

Research design adopted for this study was prototype experiment. This choice is guided by the need for flexibility in topology design and also cost consideration. Free and open source tools will be used for this study. They are widely used in research and prototyping of networks.

3.2.1 Experiment Requirements

To meet our prototype design objective, the following were identified:

1. SDN Controller – Floodlight.
2. A network emulator.
3. A traffic generator.

A basic PC with the following specification was found to be adequate for the hardware platform. Intel Core i7-5500U CPU. 2.4 GHz processor. 16GB Random Access Memory. 10/100/1000 Mbps network interface.

For software, Ubuntu Linux Operating System running on Oracle Virtual Box virtual machine was used. Linux virtual machines will be created for the SDN controller, Mininet network emulator and iPERF traffic generation and analysis tool for network performance measurements.

3.2.2 Network Setup

A topology consisting of one server and thirty four PC clients. Among the PCs we will have two acting as DDoS attackers. OpenFlow will be used for simulation of the SDN network.

The attackers will be used to simulate a protocol attack while the other machines will generate normal traffic.

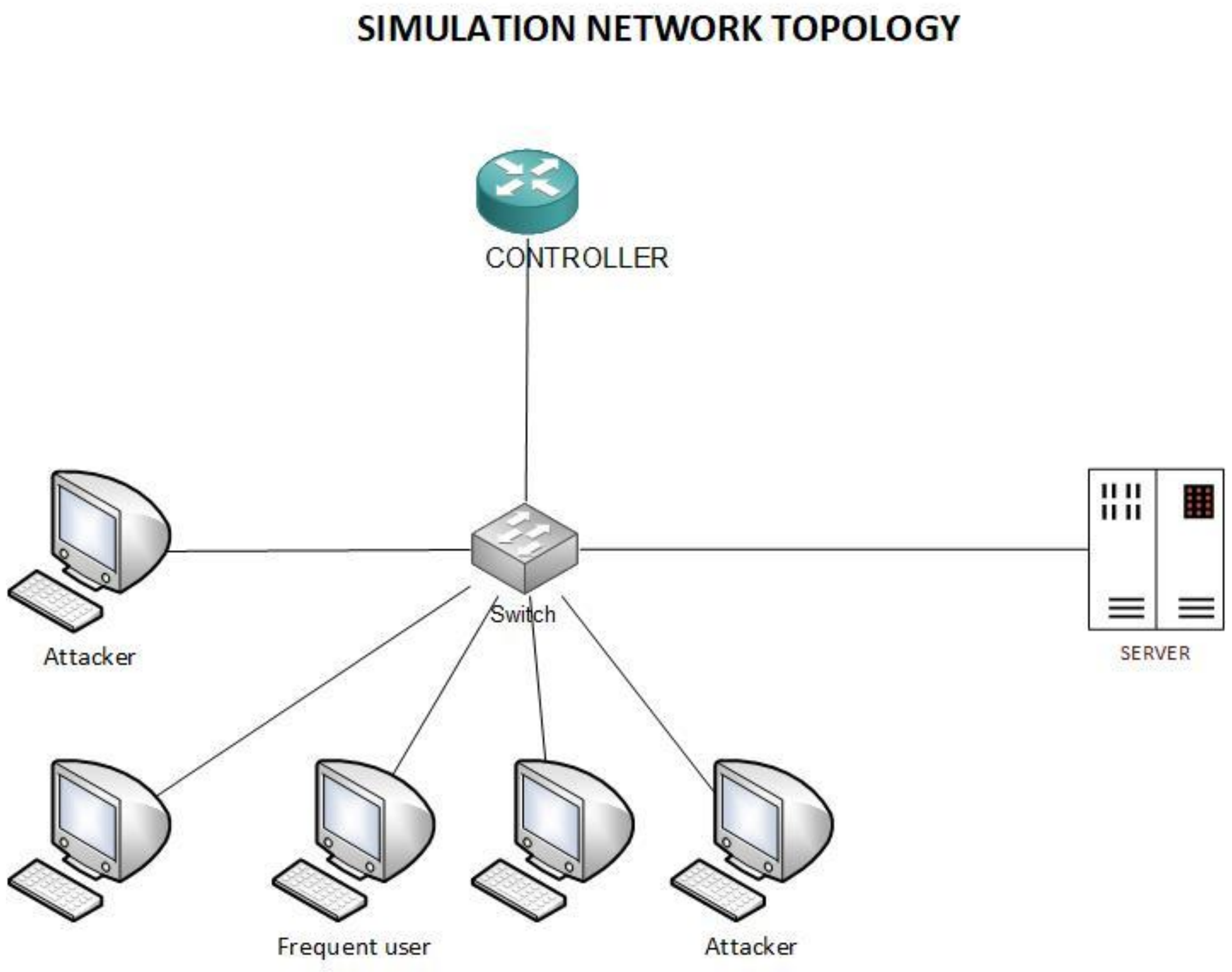


Figure 7 Simulation Network Topology

3.3 Application Development

This study and experiment will entail creating an application to run on the controller. The application is meant to detect and mitigate Ping Floods and TCP SYN DDoS attacks.

The application is a REST application built using node.js and uses JSON to communicate with the controller.

3.3.1 Software Development Model

This study will employ the classic waterfall software development because the project requirements are already clear.

3.3.2 Requirements Definition and Analysis

Requirements were identified and further classified into two categories i.e. functional requirements and non-functional requirements.

a. Functional requirements – these following functional requirements were identified:
    i. The application should be capable of examining all packets sent to the web server or of interest.
    ii. It should be capable of identifying malicious packets and dropping them and blacklisting the suspected IP addresses.
    iii. The application should be capable of communicating with the controller when installing appropriate flows in the switches depending on decisions it has made.

b. Non-functional requirements include:
    i. The application shouldn't overload the CPU and memory of the controller or machine it is running on.

3.3.3 Application design and development

The application will be developed using node.js.

Floodlight controller has a REST API through which applications can communicate with it. The main function of the application is to check all packets destined to the web server; identify the TCP packets and keep track of the SYN packets.

The application will have two main classes. The first class handles mitigation and has a total of six functions. The second class launches the application and has two functions.

Inputs/Outputs

Inputs to the application are all the packets destined to the web server. All the packets to the SDN network are also input since it has to examine all of them and determine which ones are destined to the web server, are TCP SYN packets and are destined to port 80 or port 8080.

The switches in the network collect these statistics and send them to the controller at a set interval e.g. every 2 seconds. The application can then examine these statistics and come up with appropriate action.

Output for the application is deletion of flows if an attack is detected. Otherwise no action is taken. When an attack is detected, the application should have the capability to block and black list the suspected IP to prevent further attacks. The application will allow legitimate traffic to be forwarded in all situations i.e. whether under attack or under normal situations.

3.3.4 System Testing

Functional tests are carried out throughout the development process of the application to ensure the code works as expected and debug the code. Unit testing and integration testing is also used when developing the application.

Unit Tests

Unit tests are done to ensure the various modules work as expected. The application has two classes which are tested separately to ensure they give accurate output. Stress tests were also incorporated to ensure throughput and latency of the application was acceptable.

Integration testing

Once the application is installed, integration tests are run to ensure it works as expected with the controller and the switches in the network.

3.3.5 Implementation

After testing was completed, the application was deployed on the SDN network.

3.4 Experimentation

This section gives more details about how the experiments will be done. The experiments aim at answering the research questions.

3.4.1 Fixed parameters

1. Number of nodes - 37: 1 controller, 1 web server, 2 machines to simulate an attack, 1 switch, 33 machines to simulate normal users.
2. Platform memory – 16Gb
3. Platform CPU – Intel Core i7-7500U 2.70GHz 2.90GHz
4. Traffic generated for the test, both normal and malicious traffic.

3.4.2 Variable parameters

1. The protection/defence mechanism. First, the experiments will be run without the defence mechanism in place thus allowing the controller to forward all the packets received on the network. Then the defence mechanism is set up on the controller and the experiments run again.

There will be legitimate users on the network. Each will send a GET-request continuously to a given server in the network periodically.

3.5 Research Tools

This section gives more information about how the research will be conducted and the data collection methods used. Selection of information sources is also discussed. How the data will be analyzed and evaluated is also discussed.

A case study of mitigating DDoS attacks on SDN will be carried out. Protocol based DDoS attacks and Application based DDoS attacks will be the subject of study. Our method of combating DDoS will be simulated in a virtual environment.

3.5.1 Controller

For our research we will use Floodlight because it can handle larger flows. Floodlight is compatible with all the major operating systems.

3.5.2 Network Emulator

This research will use Mininet (Mininet, 2017) as the network emulator. Mininet is a common network emulation application used with Software Defined Networks. It can emulate a whole network on a single laptop or PC using what we call kernel namespaces to avail individualistic processes having their own e.g. networking interface, address resolution protocols, route table etc.

Mininet makes use of process based virtualization to simulate switches and various hosts or machines on a given kernel. This gives us the ability to simulate large networks and even various topologies.

To create a network using Mininet, the command *mn* is used.

3.5.3 Open vSwitch

OpenvSwitch (OVS) performs multilayer switching and can be programmatically controlled by SDN controllers, this switching platform will be used in this study because it works will with both the Floodlight controller and Mininet.

OVS architecture is shown below:

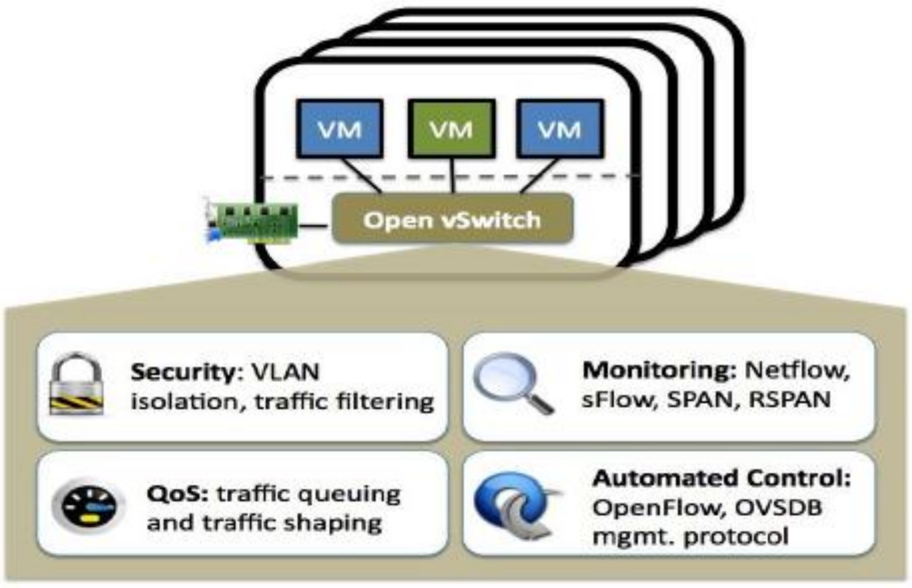

VM
VM
VM
Open vSwitch
Security: VLAN isolation, traffic filtering
Monitoring: Netflow, sFlow, SPAN, RSPAN
QoS: traffic queuing and traffic shaping
Automated Control: OpenFlow, OVSDB mgmt. protocol

Figure 8 OpenvSwitch

3.5.4 Packet Generation

Packet generation is done hping3 and nping. Hping3 has no graphical interface and is used as an OSI TCP-IP packet assembly and analyzation. In addition to ICMP pings, this tool can work on a variety of protocols such as TCP, User datagram protocol, raw internet protocol packets. Moreover, it contains the tracert mode and a unique ability to transport files over a secure channel etc.

On the other hand, NPING is a free tool used to generate packets, analyze responses, and measure response times. This tool is able to generate packets based on quite a big variety of protocols thus enabling users to have admin control of protocol headers.

Nping has a variety of functions e.g. simple ping, packet generation, stress testing, simulation of attacks such as DoS, traceroutes etc.

3.5.5 Hypervisor

Virtualbox hypervisor is available for free. It is owned by Oracle. This hypervisor may be installed on any operating system. Virtualbox will support all OSs on the guest virtual machine. (Virtualbox, 2018)

3.5.6 sFlow-RT

This project uses sFlow-RT to obtain real time visibility in terms of traffic and statistics, of the Software Defined Network. sFlow enables development of applications that are aware of the network's performance at any one time. This allows us to create applications with capability for DDoS detection and prevention, load balancing etc.

sFlow-RT has an analytics engine that obtains telemetry streams continuously from agents known as sFlow agents. These agents are found in network equipment, hosts and also apps. The agents convert this telemetry streams into metrics that can be acted upon and accessed via the REST API. This REST API enables administrators to define measurements, obtain given metrics, define thresholds, and get notifications. Applications can then be built to use this information as long as the language used supports HTTP/REST calls.

sFlow-RT enables administrators or users to have a view of applications, servers and network performance. This is achieved through a combination of network, host and app monitoring within an integrated analytics pipeline. (inMon, 2018).

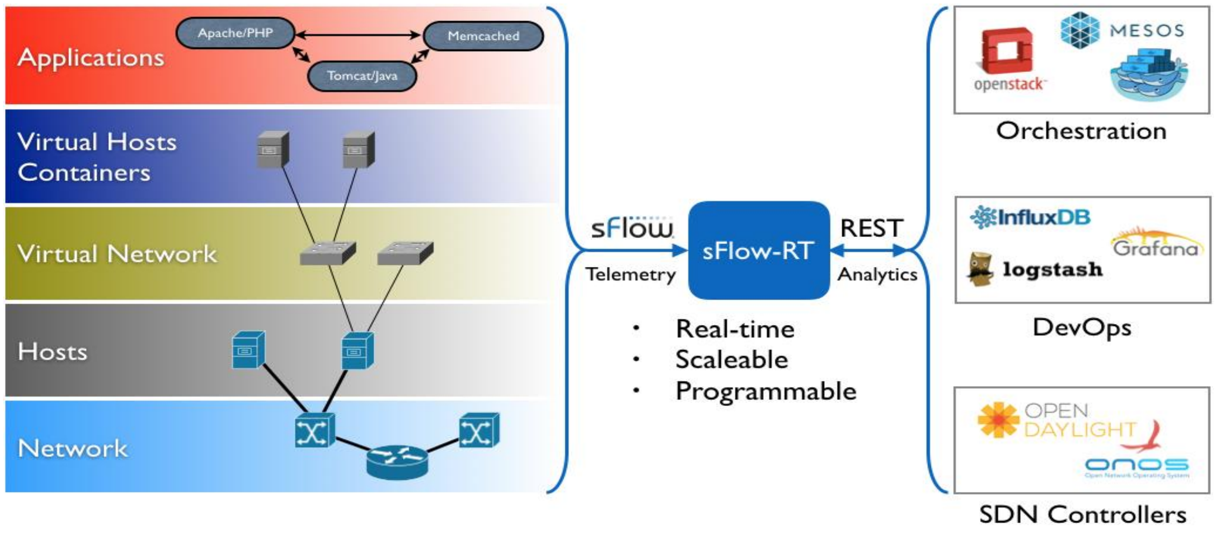
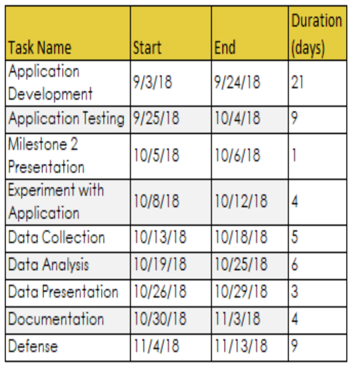


Figure 9 sFlow-RT Operation

3.5.7 OpenFlow

OpenFlow(OF) is a pioneer SDN standard that defines communication protocol enabling the controller to interact with the data plane of devices in the infrastructure layer of the SDN. SDN controllers can send configuration changes down to switches and routers using OpenFlow. Flow tables of the switches are modified by the controller using OpenFlow.

3.6 Research Schedule



Task Name	Start	End	Duration (days)
Application Development	9/3/18	9/24/18	21
Application Testing	9/25/18	10/4/18	9
Milestone 2 Presentation	10/5/18	10/6/18	1
Experiment with Application	10/8/18	10/12/18	4
Data Collection	10/13/18	10/18/18	5
Data Analysis	10/19/18	10/25/18	6
Data Presentation	10/26/18	10/29/18	3
Documentation	10/30/18	11/3/18	4
Defense	11/4/18	11/13/18	9

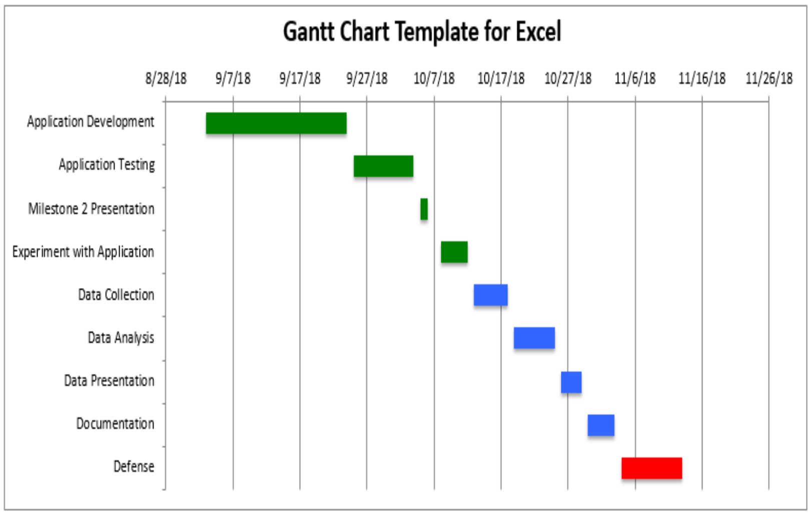


Figure 10 Project Schedule

3.7 Experimenting with the setup

Below is the experimental setup.

Topology:

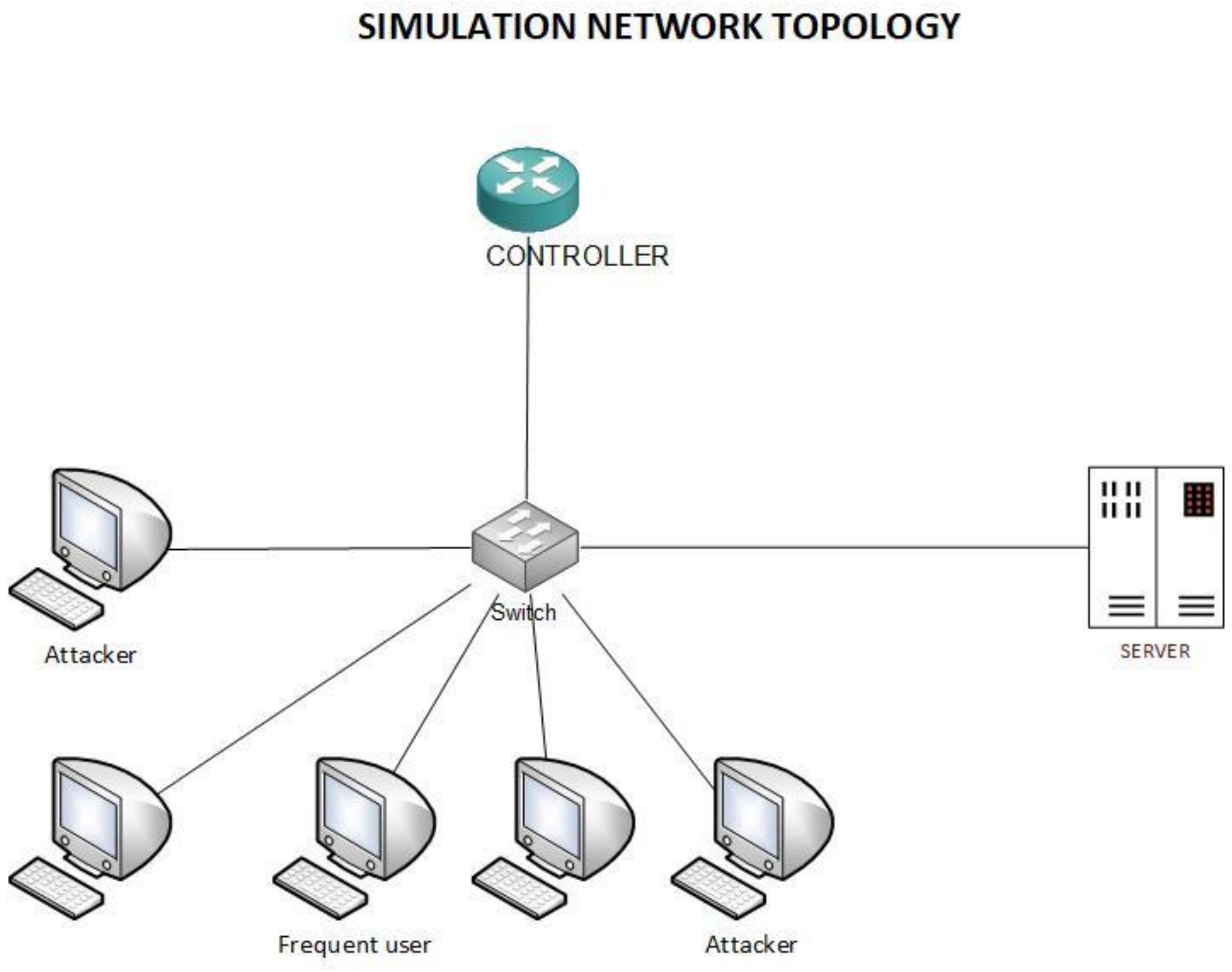


Figure 11 Simulation Network Topology

The above diagram shows the topology to be used. For our experiment, we will have 35 hosts. One of the machines will be a web server which we will use to simulate the attacks. The other hosts will be used to generate traffic on the network and also to simulate DDoS attacks on the network. The topology has one mininet switch. We also have one controller based on Floodlight. Mininet will be used to simulate the network i.e. the switch and the hosts. The controller is simulated on its own using Floodlight. The two are then integrated to work together.

For collection of statistics on the network, sFlow is configured. This tool is greatly used to demonstrate what is happening on the network. For our research we integrate sFlow and mininet on startup so that any packet flowing through our simulated network is captured. sFlow also enables us to graph traffic and protocols types on our network. This will help with interpretation of the results.

Traffic on the network is simulated using nping and ping commands.

4 CHAPTER 4 RESULTS AND DISCUSSION

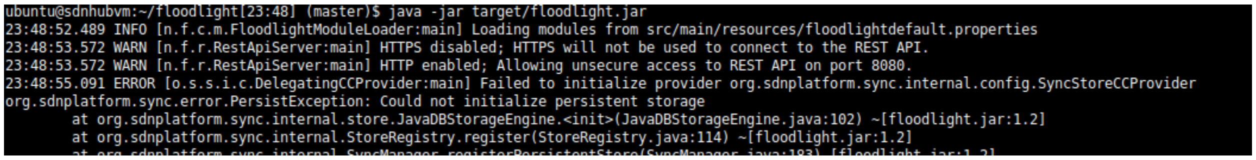
4.1 Experiment Results

T Results from experiments conducted will be discussed hence forth.

4.2 Starting Floodlight

The Floodlight controller service is started by navigating to the floodlight installation directory - /home/Ubuntu/floodlight and executing the command:

Java –jar target/floodlight.jar.



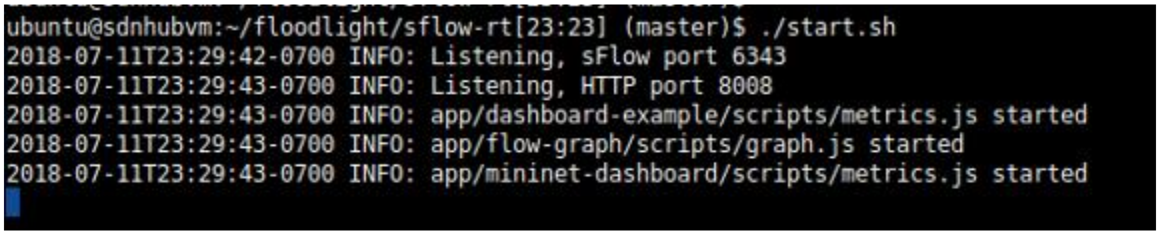
```
ubuntu@sdnhubvm:~/floodlight[23:48] (master)$ java -jar target/floodlight.jar
23:48:52.489 INFO [n.f.c.m.FloodlightModuleLoader:main] Loading modules from src/main/resources/floodlightdefault.properties
23:48:53.572 WARN [n.f.r.RestApiServer:main] HTTPS disabled; HTTPS will not be used to connect to the REST API.
23:48:53.572 WARN [n.f.r.RestApiServer:main] HTTP enabled; Allowing unsecure access to REST API on port 8080.
23:48:55.091 ERROR [o.s.s.i.c.DelegatingCCProvider:main] Failed to initialize provider org.sdnplatform.sync.internal.config.SyncStoreCCProvider
org.sdnplatform.sync.error.PersistException: Could not initialize persistent storage
        at org.sdnplatform.sync.internal.store.JavaDBStorageEngine.<init>(JavaDBStorageEngine.java:102) ~[floodlight.jar:1.2]
        at org.sdnplatform.sync.internal.StoreRegistry.register(StoreRegistry.java:114) ~[floodlight.jar:1.2]
        at org.sdnplatform.sync.internal.SyncManager.registerPersistentStore(SyncManager.java:183) [floodlight.jar:1.2]
```

Figure 12 Starting Floodlight Controller

4.3 Starting sFlow

To start sFlow, navigate to the working directory for sFlow - /home/Ubuntu/floodlight/sflow-rt and run the following command:

./start.sh



```
ubuntu@sdnhubvm:~/floodlight/sflow-rt[23:23] (master)$ ./start.sh
2018-07-11T23:29:42-0700 INFO: Listening, sFlow port 6343
2018-07-11T23:29:43-0700 INFO: Listening, HTTP port 8008
2018-07-11T23:29:43-0700 INFO: app/dashboard-example/scripts/metrics.js started
2018-07-11T23:29:43-0700 INFO: app/flow-graph/scripts/graph.js started
2018-07-11T23:29:43-0700 INFO: app/mininet-dashboard/scripts/metrics.js started
```

Figure 13 Starting sFlow

4.4 Starting Mininet

For this project, a topology file – MyTopology.py was created to allow for faster running of mininet with our preferred topology.

```
"""
Project Topology
"""
from mininet.topo import Topo
class MinimalTopo( Topo ):
    "Minimal topology with a single switch and six hosts"
```

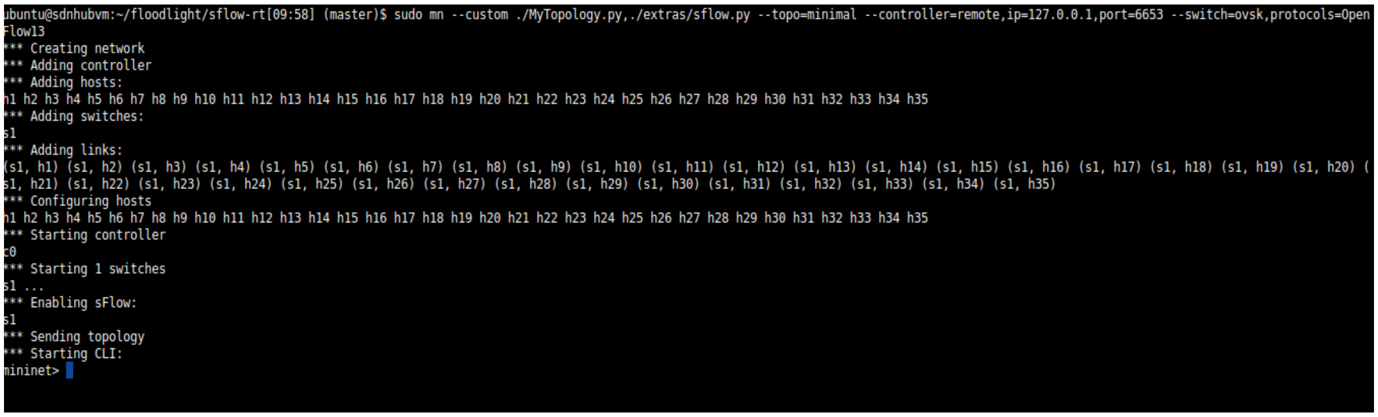
```
def build( self ):
    # Create six hosts.
    h1 = self.addHost( 'h1' )
    h2 = self.addHost( 'h2' )
    h3 = self.addHost( 'h3' )
    h4 = self.addHost( 'h4' )
    h5 = self.addHost( 'h5' )
    h6 = self.addHost( 'h6' )
    h7 = self.addHost( 'h7' )
    h8 = self.addHost( 'h8' )
    h9 = self.addHost( 'h9' )
    h10 = self.addHost( 'h10' )
    h11 = self.addHost( 'h11' )
    h12 = self.addHost( 'h12' )
    h13 = self.addHost( 'h13' )
    h14 = self.addHost( 'h14' )
    h15 = self.addHost( 'h15' )
    h16 = self.addHost( 'h16' )
    h17 = self.addHost( 'h17' )
    h18 = self.addHost( 'h18' )
    h19 = self.addHost( 'h19' )
    h20 = self.addHost( 'h20' )
    h21 = self.addHost( 'h21' )
    h22 = self.addHost( 'h22' )
    h23 = self.addHost( 'h23' )
    h24 = self.addHost( 'h24' )
    h25 = self.addHost( 'h25' )
    h26 = self.addHost( 'h26' )
    h27 = self.addHost( 'h27' )
    h28 = self.addHost( 'h28' )
    h29 = self.addHost( 'h29' )
```

```
h30 = self.addHost( 'h30' )
h31 = self.addHost( 'h31' )
h32 = self.addHost( 'h32' )
h33 = self.addHost( 'h33' )
h34 = self.addHost( 'h34' )
h35 = self.addHost( 'h35' )
# Create a switch
s1 = self.addSwitch('s1' )
# Add links between the switch and each host
self.addLink(s1,h1)
self.addLink(s1,h2)
self.addLink(s1,h3)
self.addLink(s1,h4)
self.addLink(s1,h5)
self.addLink(s1,h6)
self.addLink(s1,h7)
self.addLink(s1,h8)
self.addLink(s1,h9)
self.addLink(s1,h10)
self.addLink(s1,h11)
self.addLink(s1,h12)
self.addLink(s1,h13)
self.addLink(s1,h14)
self.addLink(s1,h15)
self.addLink(s1,h16)
self.addLink(s1,h17)
self.addLink(s1,h18)
self.addLink(s1,h19)
self.addLink(s1,h20)
self.addLink(s1,h21)
self.addLink(s1,h22)
```

```
        self.addLink(s1,h23)
        self.addLink(s1,h24)
        self.addLink(s1,h25)
        self.addLink(s1,h26)
        self.addLink(s1,h27)
        self.addLink(s1,h28)
        self.addLink(s1,h29)
        self.addLink(s1,h30)
        self.addLink(s1,h31)
        self.addLink(s1,h32)
        self.addLink(s1,h33)
        self.addLink(s1,h34)
        self.addLink(s1,h35)
# Allows the file to be imported using `mn --custom <filename> --topo minimal`
topos = {
        'minimal': MinimalTopo
 }
```

To start mininet while enabling sFlow on the switches at the same time, the command below is eecuted from the sFlow directory:

```
ubuntu@sdnhubvm:~/floodlight/sflow-rt[23:29] (master)$ sudo mn --custom
./MyTopology.py,./extras/sflow.py --topo=minimal --controller=remote,ip=127.0.0.1,port=6653
--switch=ovsk,protocols=OpenFlow13
```



```
ubuntu@sdnhubvm:~/floodlight/sflow-rt[09:58] (master)$ sudo mn --custom ./MyTopology.py,./extras/sflow.py --topo=minimal --controller=remote,ip=127.0.0.1,port=6653 --switch=ovsk,protocols=Open
Flow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4) (s1, h5) (s1, h6) (s1, h7) (s1, h8) (s1, h9) (s1, h10) (s1, h11) (s1, h12) (s1, h13) (s1, h14) (s1, h15) (s1, h16) (s1, h17) (s1, h18) (s1, h19) (s1, h20) (
s1, h21) (s1, h22) (s1, h23) (s1, h24) (s1, h25) (s1, h26) (s1, h27) (s1, h28) (s1, h29) (s1, h30) (s1, h31) (s1, h32) (s1, h33) (s1, h34) (s1, h35)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Enabling sFlow:
s1
*** Sending topology
*** Starting CLI:
mininet>
```

Figure 14 Starting mininet

4.5 Configuring h1 as a web server

To configure host h1 as the web server in the topology, run the command below from the mininet command prompt:

```
h1 python -m SimpleHTTPServer 80 &
```

```
mininet> h1 python -m SimpleHTTPServer 80 &
mininet>
```

Accessing web server:

To access the web server from host h2, use the command below:

```
h2 wget -O - h1
```

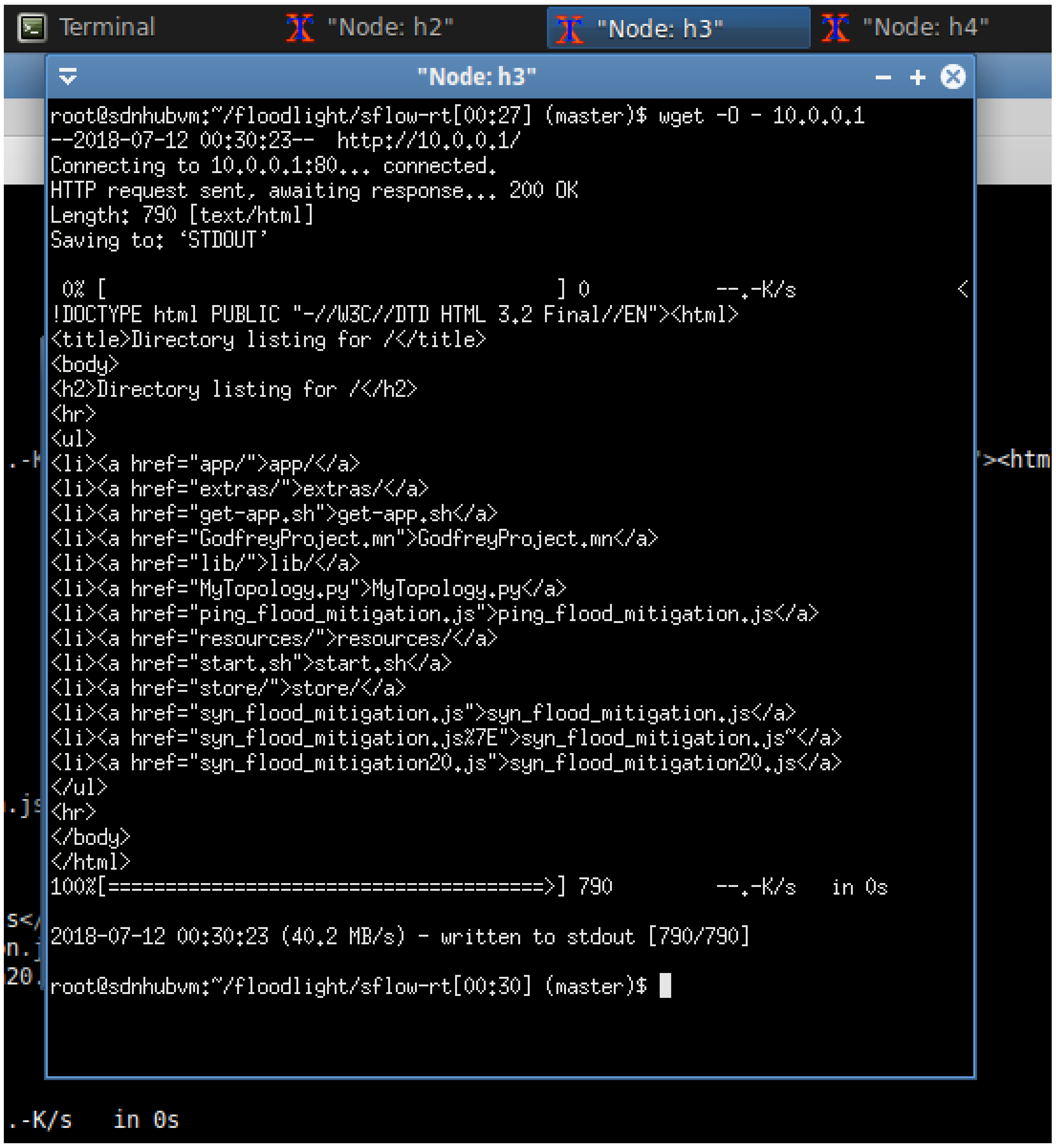
The same command can be run from h2's terminal as:

```
wget -O – 10.0.0.1
```

```
mininet> h2 wget -O - h1
--2018-07-12 00:29:19-- http://10.0.0.1/
```

```
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 790 [text/html]
Saving to: ‘STDOUT’

0% [ ] 0 --.-K/s <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href="app/">app/</a>
<li><a href="extras/">extras/</a>
<li><a href="get-app.sh">get-app.sh</a>
<li><a href="GodfreyProject.mn">GodfreyProject.mn</a>
<li><a href="lib/">lib/</a>
<li><a href="MyTopology.py">MyTopology.py</a>
<li><a href="ping_flood_mitigation.js">ping_flood_mitigation.js</a>
<li><a href="resources/">resources/</a>
<li><a href="start.sh">start.sh</a>
<li><a href="store/">store/</a>
<li><a href="syn_flood_mitigation.js">syn_flood_mitigation.js</a>
<li><a href="syn_flood_mitigation.js%7E">syn_flood_mitigation.js~</a>
<li><a href="syn_flood_mitigation20.js">syn_flood_mitigation20.js</a>
</ul>
<hr>
</body>
</html>
100%[=======================================>] 790 --.-K/s in 0s

2018-07-12 00:29:19 (55.0 MB/s) - written to stdout [790/790]
```

mininet>



```
root@sdnhubvm:~/floodlight/sflow-rt[00:27] (master)$ wget -O - 10.0.0.1
--2018-07-12 00:30:23--  http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 790 [text/html]
Saving to: 'STDOUT'

 0% [                                       ] 0           --.-K/s              <
!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href="app/">app/</a>
<li><a href="extras/">extras/</a>
<li><a href="get-app.sh">get-app.sh</a>
<li><a href="GodfreyProject.mn">GodfreyProject.mn</a>
<li><a href="lib/">lib/</a>
<li><a href="MyTopology.py">MyTopology.py</a>
<li><a href="ping_flood_mitigation.js">ping_flood_mitigation.js</a>
<li><a href="resources/">resources/</a>
<li><a href="start.sh">start.sh</a>
<li><a href="store/">store/</a>
<li><a href="syn_flood_mitigation.js">syn_flood_mitigation.js</a>
<li><a href="syn_flood_mitigation.js%7E">syn_flood_mitigation.js~</a>
<li><a href="syn_flood_mitigation20.js">syn_flood_mitigation20.js</a>
</ul>
<hr>
</body>
</html>
100%[======================================>] 790         --.-K/s    in 0s

2018-07-12 00:30:23 (40.2 MB/s) - written to stdout [790/790]

root@sdnhubvm:~/floodlight/sflow-rt[00:30] (master)$
```

Figure 15 Accessing webserver

4.6 Simulating traffic on the network

Nping was used to simulate traffic on the virtual network. The following commands were used:

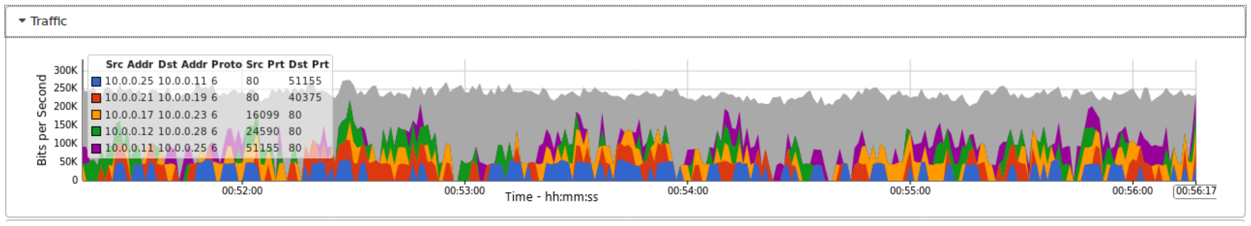
/* Send UDP packets with 100 bytes of random data */

```
h7 - nping --udp 10.0.0.35 -p 53 --data-length 100 -c 50000000
h10 - nping --udp 10.0.0.29 -p 53 --data-length 100 -c 10000

/* Send ICMP packets */
h8 - ping 10.0.0.30 -c 20000000
h9 - ping 10.0.0.22 -c 20000000
# Send TCP packets at a rate of 100pkts/sec
h11 - sudo nping --tcp 10.0.0.25 --rate 100 -c 10000000
h12 - sudo nping --tcp 10.0.0.28 --rate 100 -c 10000000
h13 - sudo nping --tcp 10.0.0.27 --rate 100 -c 1000000
h14 - sudo nping --tcp 10.0.0.26 --rate 100 -c 20000000
h15 - sudo nping --tcp 10.0.0.25 --rate 100 -c 30000000
h16 - sudo nping --tcp 10.0.0.24 --rate 100 -c 20000000
h17 - sudo nping --tcp 10.0.0.23 --rate 100 -c 20000000
h18 - sudo nping --tcp 10.0.0.22 --rate 100 -c 20000000
h19 - sudo nping --tcp 10.0.0.21 --rate 100 -c 20000000
h20 - sudo nping --tcp 10.0.0.20 --rate 100 -c 20000000

# Send ARP requests to 10.0.0.0/24 subnet
h21 - sudo nping --arp 10.0.0.0/24 --delay 100ms
```

4.7 Ping Flood Simulation:

4.7.1 Normal traffic:

Traffic on the network before attack:



	Src Addr	Dst Addr	Proto	Src Prt	Dst Prt
🟦	10.0.0.25	10.0.0.11	6	80	51155
🟥	10.0.0.21	10.0.0.19	6	80	40375
🟧	10.0.0.17	10.0.0.23	6	16099	80
🟩	10.0.0.12	10.0.0.28	6	24590	80
🟪	10.0.0.11	10.0.0.25	6	51155	80

Figure 16 Normal traffic

Figure 16 shows normal traffic on the virtual network. This is before the ping flood attack is simulated. This traffic is generated by the commands nping commands shown previously.
The different colors indicate traffic from one host to the other e.g.
Blue: ICMP traffic from 10.0.0.25 to 10.0.0.11.
Red: ICMP traffic from 10.0.0.21 to 10.0.0.19
Orange: ICMP traffic from 10.0.0.17 to 10.0.0.23
Green: ICMP traffic from 10.0.0.12 to 10.0.0.28
Maroon: ICMP traffic from 10.0.0.11 to 10.0.0.25

The graph shows normal conditions and traffic as per our simulation. Hosts exchanging traffic mostly via ICMP among themselves.
At this point the attack has not started yet. The switch has flow rules installed by the controller that allow traffic to be exchanged among all hosts.

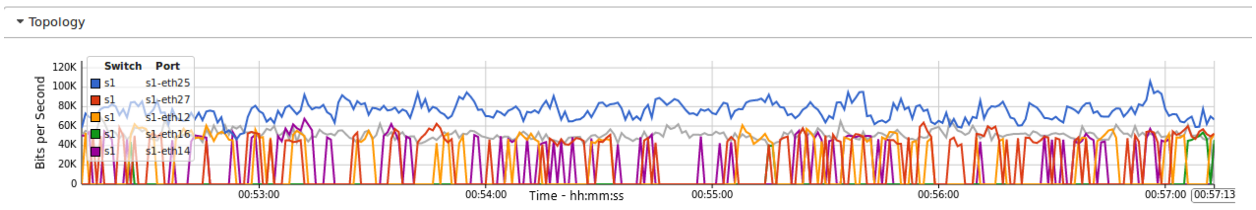


Figure 17Normal topology traffic

Figure 17 shows normal topology traffic on the simulated network. This is generated by the nping commands to simulate normal traffic.
The graph indicates traffic as seen on the mininet switch in our topology. Different colors are used for each host connected on a port on the switch.
Blue: Traffic on port 25 of the switch for host 10.0.0.25
Red: Traffic on port 27 of the switch for host 10.0.0.27
Yellow: Traffic on port 12 of the switch for host 10.0.0.12
Green: Traffic on port 16 of the switch for host 10.0.0.16
Maroon: Traffic on port 14 of the switch for host 10.0.0.14

This is traffic generated by the nping commands shown previously.

All traffic is allowed since the flow tables have not yet blocked any IP addresses. The SDN controller installed the flow tables in the switch which allow this traffic to go through.

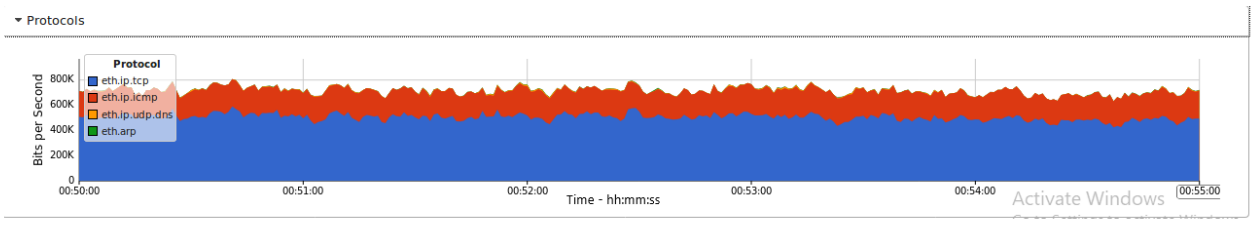


Figure 18 Normal protocol traffic

Figure 18 shows protocols running on the network. TCP traffic is the most common with ICMP, UDP and ARP as shown above.

The traffic was generated by nping commands on the various hosts. Different protocols on the network have been represented by different colors on the graph. The traffic running on the network represented by different colors is:

Blue: IP TCP traffic on the network.

Red: IP ICMP traffic.

Yellow: IP UDP DNS traffic.

Green: ARP traffic.

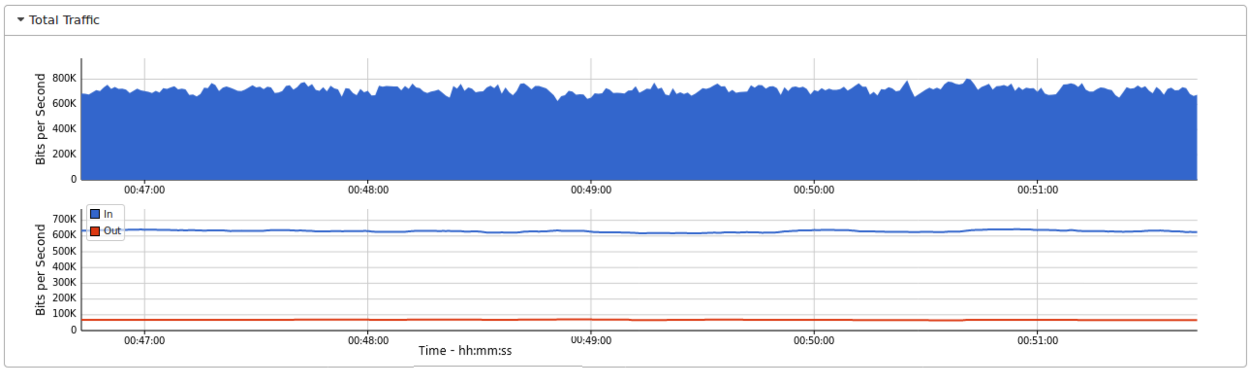
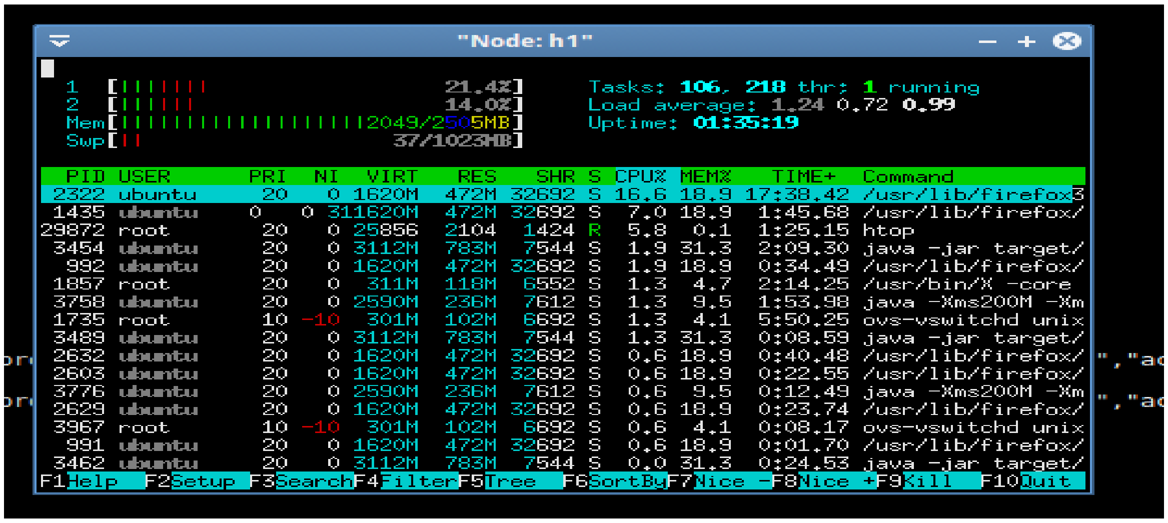


Figure 19 Normal traffic

Figure 19 shows all the total traffic on the simulated network under normal conditions.

The top graph shows total traffic which is just below 800kbps and is shown in blue graph. On the lower graph, incoming traffic is shown using a blue line at just over 600kbps and outgoing traffic is shown using a red line and is just below 100kbps.

4.7.2 CPU Usage on Server under normal traffic conditions:

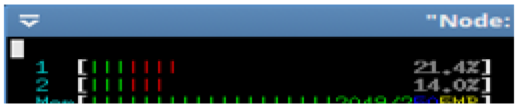


```
"Node: h1"

  1   [|||||||                     21.4%]     Tasks: 106, 218 thr: 1 running
  2   [||||||                      14.0%]     Load average: 1.24 0.72 0.99
  Mem[||||||||||||||||||||2049/2505MB]     Uptime: 01:35:19
  Swp[||                         37/1023MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 2322 ubuntu     20   0 1620M  472M 32692 S 16.6 18.9 17:38.42 /usr/lib/firefox3
 1435 ubuntu    0     0 311620M 472M 32692 S  7.0 18.9  1:45.68 /usr/lib/firefox/
29872 root       20   0 25856  2104  1424 R  5.8  0.1  1:25.15 htop
 3454 ubuntu     20   0 3112M  783M  7544 S  1.9 31.3  2:09.30 java -jar target/
  992 ubuntu     20   0 1620M  472M 32692 S  1.9 18.9  0:34.49 /usr/lib/firefox/
 1857 root       20   0  311M  118M  6552 S  1.3  4.7  2:14.25 /usr/bin/X -core
 3758 ubuntu     20   0 2590M  236M  7612 S  1.3  9.5  1:53.98 java -Xms200M -Xm
 1735 root       10 -10  301M  102M  6692 S  1.3  4.1  5:50.25 ovs-vswitchd unix
 3489 ubuntu     20   0 3112M  783M  7544 S  1.3 31.3  0:08.59 java -jar target/
 2632 ubuntu     20   0 1620M  472M 32692 S  0.6 18.9  0:40.48 /usr/lib/firefox/
 2603 ubuntu     20   0 1620M  472M 32692 S  0.6 18.9  0:22.55 /usr/lib/firefox/
 3776 ubuntu     20   0 2590M  236M  7612 S  0.6  9.5  0:12.49 java -Xms200M -Xm
 2629 ubuntu     20   0 1620M  472M 32692 S  0.6 18.9  0:23.74 /usr/lib/firefox/
 3967 root       10 -10  301M  102M  6692 S  0.6  4.1  0:08.17 ovs-vswitchd unix
  991 ubuntu     20   0 1620M  472M 32692 S  0.6 18.9  0:01.70 /usr/lib/firefox/
 3462 ubuntu     20   0 3112M  783M  7544 S  0.0 31.3  0:24.53 java -jar target/
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figure 20 CPU usage on server

Figure 20 shows the CPU usage on the web server under normal conditions. The figure shows a value of 16.6%.

Htop command was used to check resource usage on the web server. The command helps administrators monitor processes running on the system along with their full command lines.



```
"Node:

  1   [|||||||                     21.4%]
  2   [||||||                      14.0%]
```

Figure 21 htop CPU output

With regards to the above screenshot, numbers 1 to 2 are CPU cores for the web server and the progress bars next to them describe their usage. The progress bars contain multiple colors. The colors have the following meanings:

Blue: Shows the percentage of CPU used by low priority processes. (nice > 0)

Green: Indicates percentage of CPU used for processes owned by normal users.

Red: Displays percentage of CPU used by system processes.

Orange: Shows percentage of CPU used by IRQ time.

Magenta: Indicates percentage of CPU consumed by Soft IRQ time.

Grey: Percentage of CPU consumed by IO Wait time.

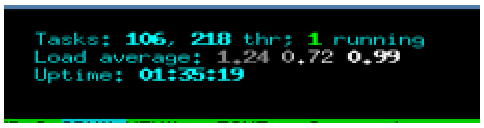
Cyan: Percentage of CPU consumed by Steal time.

Just below the CPU stats we have information on memory and swap usage. A similar progress bar with different colors is used. The colors have the following meanings:

Green - percentage of RAM utilized by memory pages

Blue - percentage of RAM used by buffer pages

Orange - percentage of RAM used by cache pages



```
Tasks: 106, 218 thr; 1 running
Load average: 1.24 0.72 0.99
Uptime: 01:35:19
```

Figure 22 Tasks, threads

Htop's output also indicates number of tasks and threads running on the system. In our case, we have 106 tasks with 218 threads but only 1 process is running.

Tasks represent all open processes. However, not each open process consumes CPU all the time. There are a number of states in which a process could exist:

R: Running – process is actively using CPU.

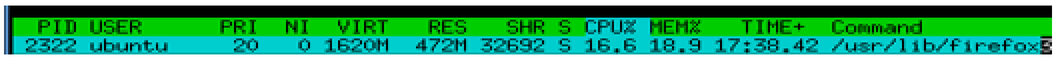
T/S: Traced/Stopped – process is currently in stopped or paused state.

Z: Zombie or defunct – process has completed execution but still has an entry in the process table.

S: Sleeping – Most common state for many processes. Generally, processes are in the sleep state for most of the time and perform small checks at a constant interval of time, or wait for user input before it comes back to running state.

The load average is the system's load average. The three values are for the last minute, last five minutes and the last 15 minutes.

Uptime value refers to system's uptime since last reboot.



```
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 2322 ubuntu     20   0 1620M  472M 32692 S 16.6 18.9 17:38.42 /usr/lib/firefox
```

Figure 23 htop detailed process information.

PID – Process ID number.

USER – process owner.

PRI – process priority as viewed by the Linux kernel.

N – Process priority reset by the user or root.

VIR – virtual memory that a process is consuming.

RES – physical memory that a process is consuming.

SHR – shared memory that a process is consuming.

S – Current state of a process.

CPU% – percentage of CPU consumed by each process.

MEM% – percentage of Memory consumed by each process.

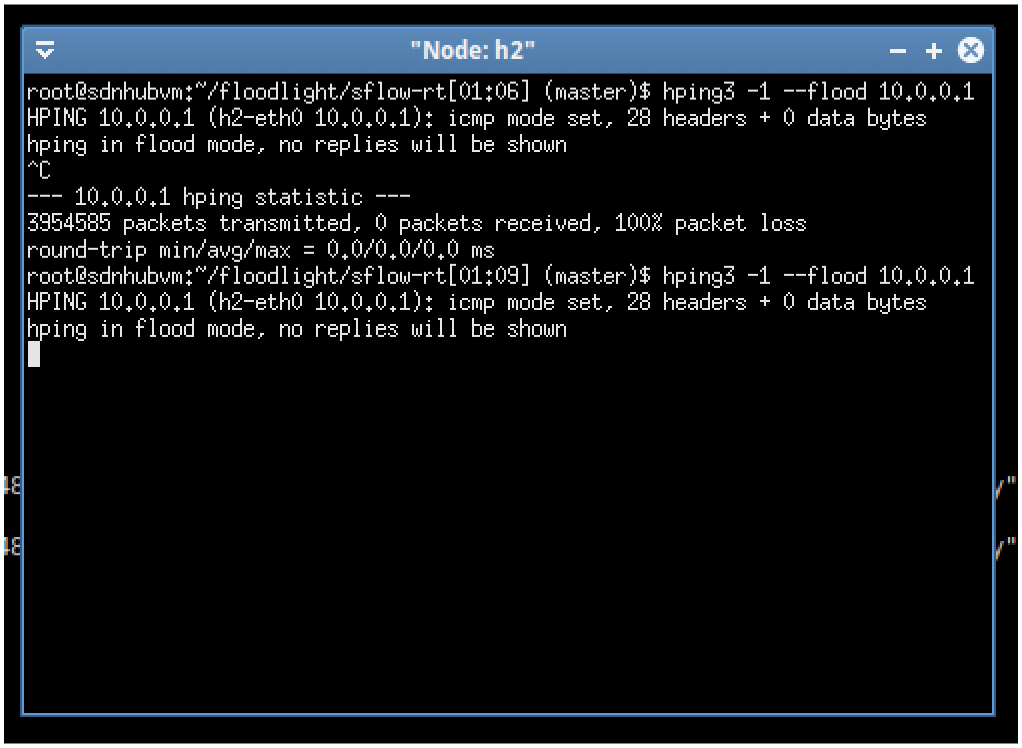
TIME+ – time since process execution has started.

Command – full command execution in parallel to each process.

At the moment the CPU is at 16.6% indicating that the system is not overloaded. Usage for CPU cores 1 and 2 are also low at 21% and 14% respectively.

4.7.3 With no Ping Flood Mitigation:

Ping Flood Attack is launched from h2 using hping3 utility.



```
root@sdnhubvm:~/floodlight/sflow-rt[01:06] (master)$ hping3 -1 --flood 10.0.0.1
HPING 10.0.0.1 (h2-eth0 10.0.0.1): icmp mode set, 28 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 10.0.0.1 hping statistic ---
3954585 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@sdnhubvm:~/floodlight/sflow-rt[01:09] (master)$ hping3 -1 --flood 10.0.0.1
HPING 10.0.0.1 (h2-eth0 10.0.0.1): icmp mode set, 28 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

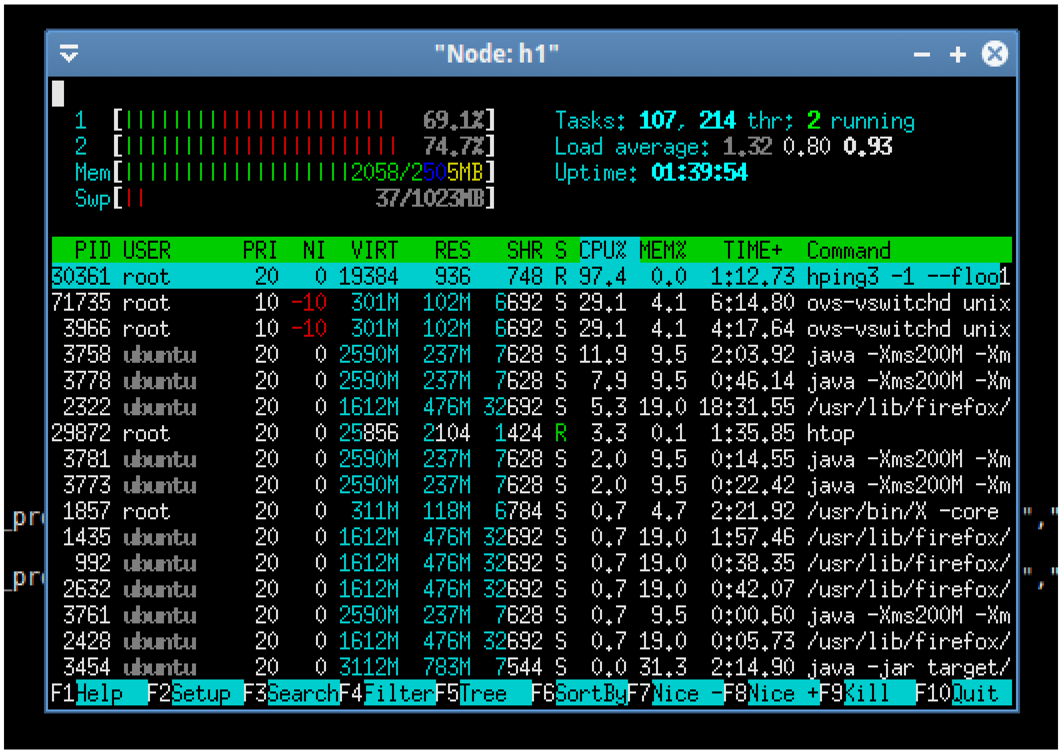
Figure 24 Ping flood attack launched

Figure 24 shows a ping flood attack initiated towards h1, the web server. The command used is hping3 -1 –flood 10.0.0.1.

10.0.0.1 is the IP address of the web server.

The attack has been sourced from host 2.

4.7.4 CPU Usage on Server:



"Node: h1"

```
  1  [|||||||||||||||||||||||       69.1%]     Tasks: 107, 214 thr; 2 running
  2  [||||||||||||||||||||||||      74.7%]     Load average: 1.32 0.80 0.93
  Mem[|||||||||||||||||||2058/2505MB]     Uptime: 01:39:54
  Swp[||                    37/1023MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
30361 root       20   0 19384   936   748 R 97.4  0.0  1:12.73 hping3 -1 --flood
71735 root       10 -10  301M  102M  6692 S 29.1  4.1  6:14.80 ovs-vswitchd unix
 3966 root       10 -10  301M  102M  6692 S 29.1  4.1  4:17.64 ovs-vswitchd unix
 3758 ubuntu     20   0 2590M  237M  7628 S 11.9  9.5  2:03.92 java -Xms200M -Xm
 3778 ubuntu     20   0 2590M  237M  7628 S  7.9  9.5  0:46.14 java -Xms200M -Xm
 2322 ubuntu     20   0 1612M  476M 32692 S  5.3 19.0 18:31.55 /usr/lib/firefox/
29872 root       20   0 25856  2104  1424 R  3.3  0.1  1:35.85 htop
 3781 ubuntu     20   0 2590M  237M  7628 S  2.0  9.5  0:14.55 java -Xms200M -Xm
 3773 ubuntu     20   0 2590M  237M  7628 S  2.0  9.5  0:22.42 java -Xms200M -Xm
 1857 root       20   0  311M  118M  6784 S  0.7  4.7  2:21.92 /usr/bin/X -core
 1435 ubuntu     20   0 1612M  476M 32692 S  0.7 19.0  1:57.46 /usr/lib/firefox/
  992 ubuntu     20   0 1612M  476M 32692 S  0.7 19.0  0:38.35 /usr/lib/firefox/
 2632 ubuntu     20   0 1612M  476M 32692 S  0.7 19.0  0:42.07 /usr/lib/firefox/
 3761 ubuntu     20   0 2590M  237M  7628 S  0.7  9.5  0:00.60 java -Xms200M -Xm
 2428 ubuntu     20   0 1612M  476M 32692 S  0.7 19.0  0:05.73 /usr/lib/firefox/
 3454 ubuntu     20   0 3112M  783M  7544 S  0.0 31.3  2:14.90 java -jar target/
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

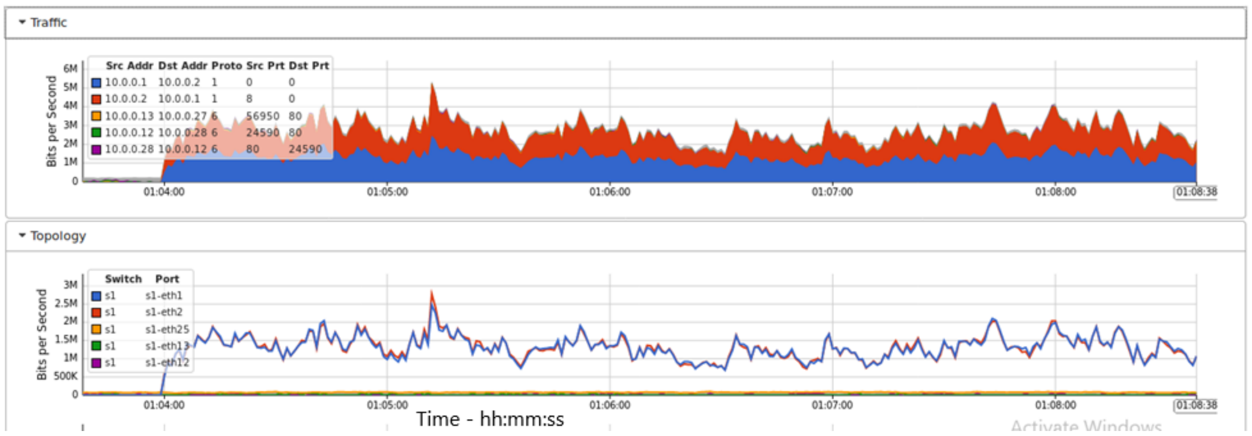
Figure 25 CPU usage on server

Figure 25 shows that the CPU usage on the web server has shot up to 97.4% during the ping flood attack.

Both CPU cores 1 and 2 have also had their usage go up to 69% and 74% respectively. Memory and swap usage has not changed much.

Tasks and threads running also remain almost the same – 107 tasks and 214 threads with only 2 processes running at the moment.

At this stage the SDN controller has no intelligence to detect the attack and stop the same. So the web server is exposed and overwhelmed thus making the services available.



	Src Addr	Dst Addr	Proto	Src Prt	Dst Prt
🟦	10.0.0.1	10.0.0.2	1	0	0
🟥	10.0.0.2	10.0.0.1	1	8	0
🟧	10.0.0.13	10.0.0.27	6	56950	80
🟩	10.0.0.12	10.0.0.28	6	24590	80
🟪	10.0.0.28	10.0.0.12	6	80	24590

	Switch	Port
🟦	s1	s1-eth1
🟥	s1	s1-eth2
🟧	s1	s1-eth25
🟩	s1	s1-eth13
🟪	s1	s1-eth12

Figure 26 Traffic on network

Figure 26 shows traffic on the network spiking up when the ping flood attack is initiated. The graph shows traffic on the network and is consistent with our simulation as it shows traffic shown in blue originating from 10.0.0.1, the web server to 10.0.0.2, the attacking machine. Red indicates traffic from the attacking machine, 10.0.0.2 to the web server, 10.0.0.1.

The lower graph shows traffic as it passes through the mininet switch. The blue line represents port 1 on the switch on which the web server is connected and the red line is for port 2 on which the attacking machine is connected.

The two graphs also indicate a spike in traffic from the normal 600kbps to almost ten times at 6Mbps.

In this instance, the SDN did not adapt to the conditions of the network i.e. detect the DDoS and stop it while allowing normal traffic to continue flowing. The malicious ping flood continued thus overwhelming the web server hence making it unresponsive. Its services are then made inaccessible. Normal users cannot access the web page because the malicious users have made it unable to respond to legitimate requests.

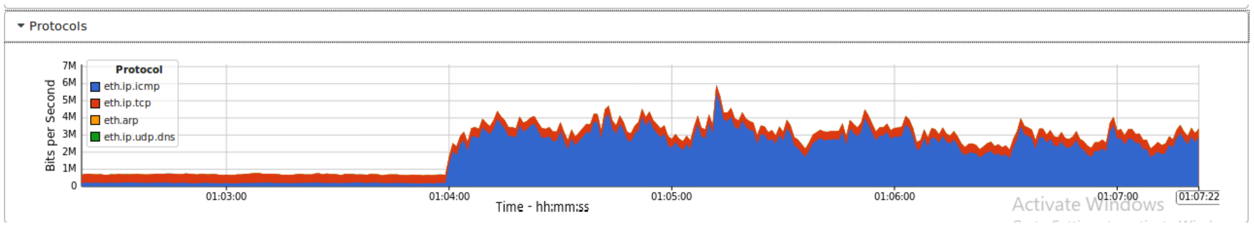


Figure 27 Protocol traffic during ping flood attack

Figure 27 shows protocol traffic on the network also increasing during the simulated attack. As indicated on the image, most of the traffic is ICMP traffic. Ping flood is based on ICMP traffic. This is shown by the blue on the graph.

Red color represents IP TCP traffic, Yellow is for ARP traffic while green indicated UDP DNS traffic on the simulated network.

The graph also shows a sudden increase in the ICMP traffic. It is at this point that the ping flood started.

The traffic continues unabated because at this moment the controller is not capable of detecting the DDoS and then stopping the same. Thus the server will end up being overwhelmed thus making the service unavailable i.e. the web service.

The SDN network still has no intelligence or an application to help it determine the legality of traffic currently traversing the network. Malicious traffic is allowed to flow through the network thus causing disruptions to other normal operations on the network. The controller does not detect the malicious traffic hence it cannot instruct the switches to drop this DDoS attack that is ongoing.

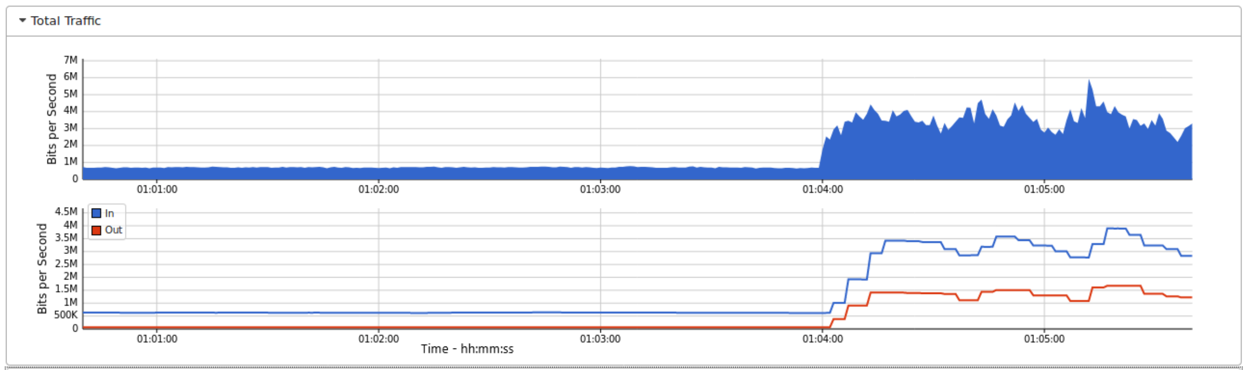
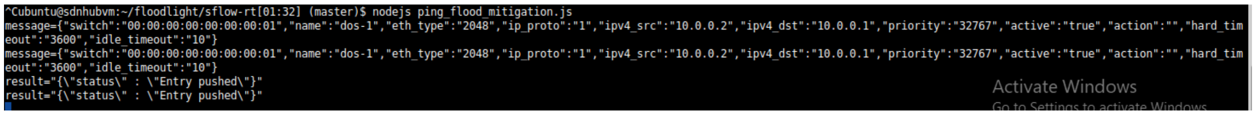


Figure 28 Traffic on network

Figure 28 is a graph of total traffic running on the network. Traffic flow was normal until when the ping flood attack was initiated leading to the traffic shooting up as shown in the image above. The lower graph shows total incoming traffic using the blue line and total outgoing traffic using the red line.

This diagram also indicates that the SDN, being unable to detect the malicious traffic, allowed total traffic to explode and go up unabated. At this point, all traffic is seen as normal traffic since it has no mechanism to differentiate between normal and malicious traffic.
The lower section of the graph also shows traffic coming in and going out increasing. The SDN allowed all traffic to come in and be responded to by the server. If this continues, it will reach a point where the server will be overwhelmed hence making it impossible to respond to legitimate queries. This is due to inability of the controller to put a stop to the attack by instructing the switches to drop the malicious traffic.

4.7.5 Running the mitigation:



```
^Cubuntu@sdnhubvm:~/floodlight/sflow-rt[01:32] (master)$ nodejs ping_flood_mitigation.js
message={"switch":"00:00:00:00:00:00:00:01","name":"dos-1","eth_type":"2048","ip_proto":"1","ipv4_src":"10.0.0.2","ipv4_dst":"10.0.0.1","priority":"32767","active":"true","action":"","hard_tim
eout":"3600","idle_timeout":"10"}
message={"switch":"00:00:00:00:00:00:00:01","name":"dos-1","eth_type":"2048","ip_proto":"1","ipv4_src":"10.0.0.2","ipv4_dst":"10.0.0.1","priority":"32767","active":"true","action":"","hard_tim
eout":"3600","idle_timeout":"10"}
result="{\"status\" : \"Entry pushed\"}"
result="{\"status\" : \"Entry pushed\"}"
```

Figure 29 Ping flood mitigation script

Figure 29 shows the mitigation script being run to detect and stop the ping flood attack.

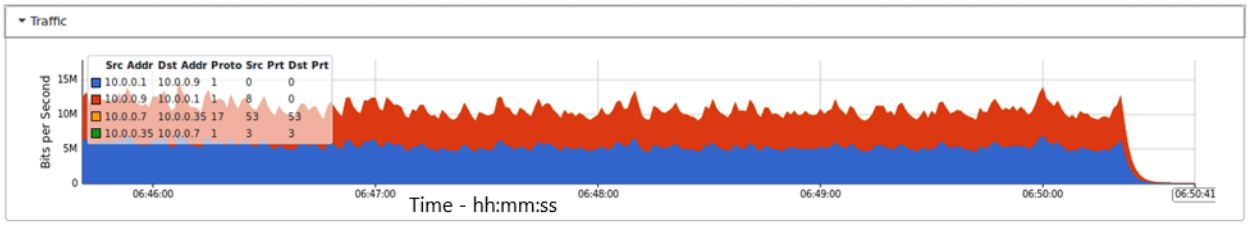


Figure 30 Traffic drop after ping flood mitigation

After the mitigation script is run, the ping flood traffic is now dropped by the switch after new flow entries are installed by the controller into the switch.

Blue indicates traffic from the web server towards the attacking machine 10.0.0.9 while red shows the reverse traffic originating from the attacking machine 10.0.0.9 destined to the web server, 10.0.0.1.

When the mitigation script is applied, the DDoS traffic drops sharply as shown in the graph. The SDN controller was able to detect the source of the malicious traffic. It then installed flows in the switch that ensure all traffic from the malicious source is dropped.

The application gives the SDN intelligence to differentiate between legitimate and malicious flows. Once malicious flows are detected, the controller installs flow rules into the switches instructing the switches to drop malicious traffic. In this case, any traffic originating from the IP address 10.0.0.9.

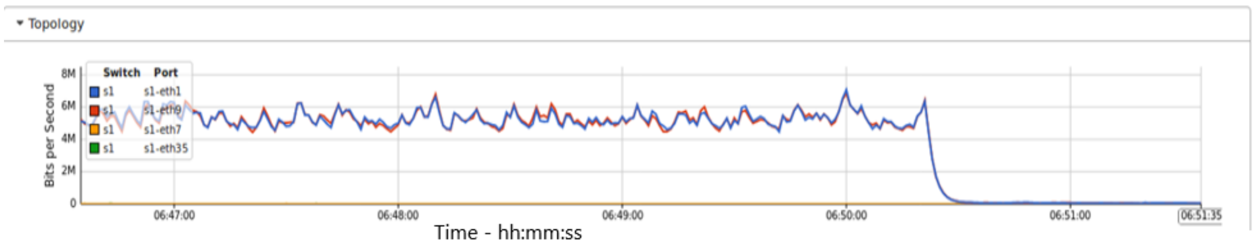


Figure 31 Topology traffic after mitigation

Figure 31 shows total topology traffic falling after the mitigation script is executed. The graph shows traffic on the switch. Blue line indicates traffic on port 1 of the switch. This is the port on which the web server is connected.

Red line shows traffic on port 9 on which the attacking host is connected.

Once the mitigation is executed, the SDN is able to detect the malicious traffic and drop the same thus eliminating the DDoS.

The sharp drop in traffic on the switch is a result of the switch dropping packets deemed malicious by the controller. When the DDoS attack was detected, the controller immediately created rules that instruct the switch to drop any packets from the malicious host. Thus ensuring that the server is safe and continues to operate optimally.

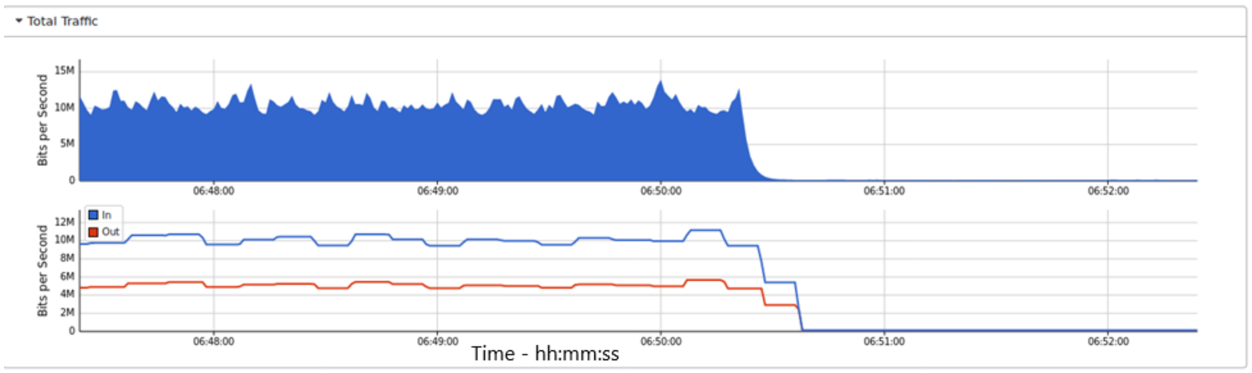


Figure 32 Total traffic after ping flood mitigation

Figure 32 shows total traffic falling after the ping flood attack has been mitigated. Most of the traffic was generated by the ping flood attack. This is shown by the graph at the top. The ping flood attack generated up to just over 10Mbps of traffic on the network.

The lower graph indicates incoming traffic using the blue line and outgoing traffic using the red line. The traffic significantly dropped after the detection mechanism was deployed. This enabled the SDN controller to detect malicious flows, install appropriate flows on the switch which made the switches drop malicious traffic thus the reduction in traffic on the network.

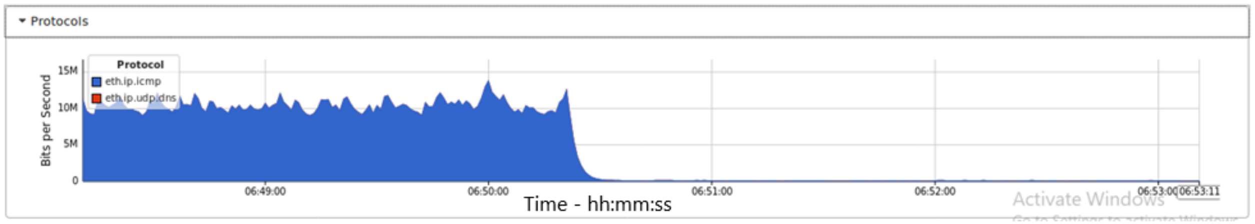
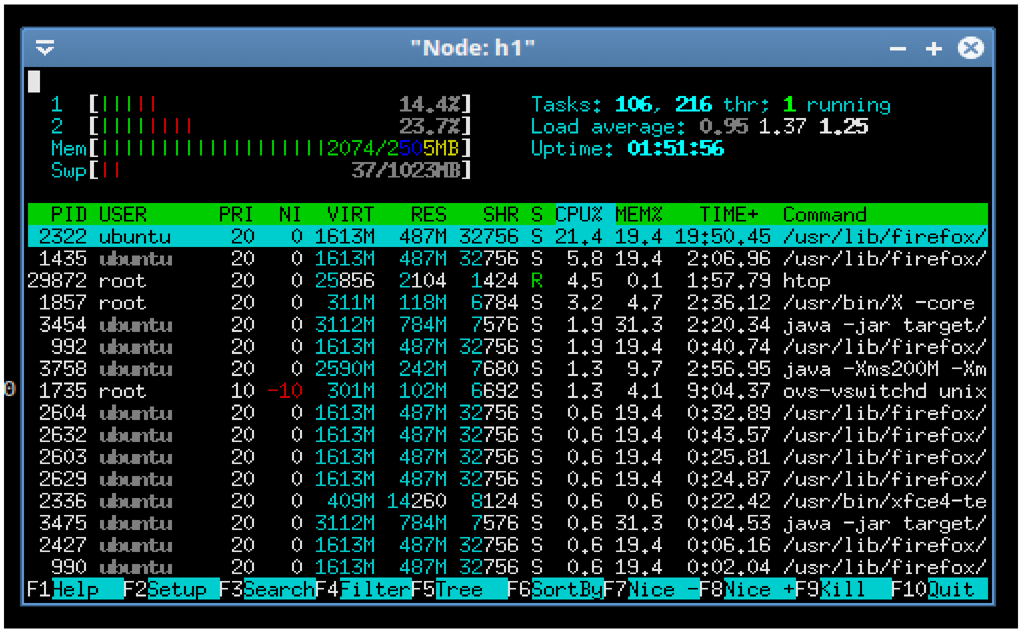


Figure 33 Protocol traffic after mitigation

The figure above shows protocol traffic which is mainly ICMP dropping after the mitigation has taken effect. Since we are simulating a ping flood attack, most of the traffic is ICMP and this is represented by the blue color on the graph. There is little UDP DNS traffic represented by the red color on the network.

Ping flood is composed of ICMP traffic, hence the SDN used its intelligence to detect the ping flood DDoS attack. It then installed appropriate flow rules onto the switches instructing the switch to drop any traffic from the malicious sources thus resulting in the sharp drop on the graph.



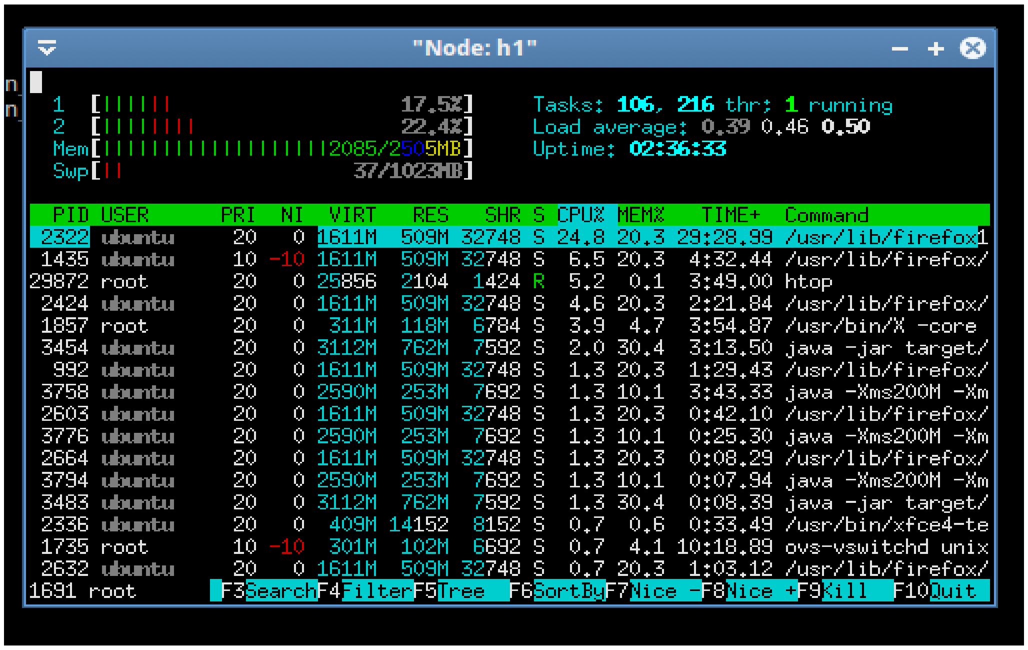
```
"Node: h1"

  1  [|||||                    14.4%]     Tasks: 106, 216 thr; 1 running
  2  [||||||||                 23.7%]     Load average: 0.95 1.37 1.25
  Mem[|||||||||||||||||||2074/2505MB]     Uptime: 01:51:56
  Swp[||                    37/1023MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 2322 ubuntu     20   0 1613M  487M 32756 S 21.4 19.4 19:50.45 /usr/lib/firefox/
 1435 ubuntu     20   0 1613M  487M 32756 S  5.8 19.4  2:06.96 /usr/lib/firefox/
29872 root       20   0 25856  2104  1424 R  4.5  0.1  1:57.79 htop
 1857 root       20   0  311M  118M  6784 S  3.2  4.7  2:36.12 /usr/bin/X -core
 3454 ubuntu     20   0 3112M  784M  7576 S  1.9 31.3  2:20.34 java -jar target/
  992 ubuntu     20   0 1613M  487M 32756 S  1.9 19.4  0:40.74 /usr/lib/firefox/
 3758 ubuntu     20   0 2590M  242M  7680 S  1.3  9.7  2:56.95 java -Xms200M -Xm
 1735 root       10 -10  301M  102M  6692 S  1.3  4.1  9:04.37 ovs-vswitchd unix
 2604 ubuntu     20   0 1613M  487M 32756 S  0.6 19.4  0:32.89 /usr/lib/firefox/
 2632 ubuntu     20   0 1613M  487M 32756 S  0.6 19.4  0:43.57 /usr/lib/firefox/
 2603 ubuntu     20   0 1613M  487M 32756 S  0.6 19.4  0:25.81 /usr/lib/firefox/
 2629 ubuntu     20   0 1613M  487M 32756 S  0.6 19.4  0:24.87 /usr/lib/firefox/
 2336 ubuntu     20   0  409M 14260  8124 S  0.6  0.6  0:22.42 /usr/bin/xfce4-te
 3475 ubuntu     20   0 3112M  784M  7576 S  0.6 31.3  0:04.53 java -jar target/
 2427 ubuntu     20   0 1613M  487M 32756 S  0.6 19.4  0:06.16 /usr/lib/firefox/
  990 ubuntu     20   0 1613M  487M 32756 S  0.6 19.4  0:02.04 /usr/lib/firefox/
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figure 34 CPU usage after mitigation

Figure 34 shows CPU usage has gone down to normal levels. Down from 97% to 21%. This is after the ping flood attack has been detected and mitigated. Usage for CPU cores 1 and 2 has also reduced to 14% and 23% respectively. Memory usage and swap usage were not affected.

The number of tasks and threads was also barely the same.

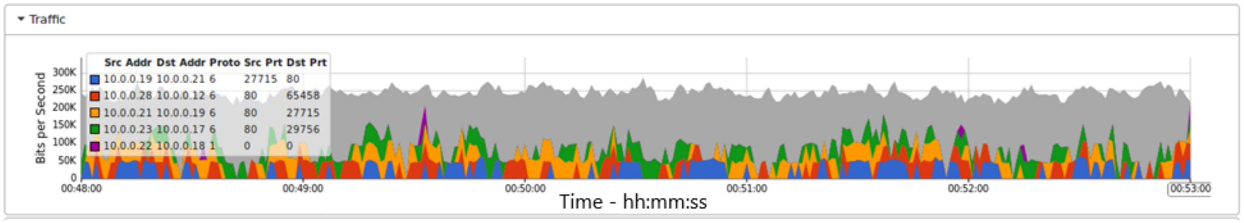
4.8 TCP SYN MITIGATION:



```
  1  [||||||                      17.5%]     Tasks: 106, 216 thr; 1 running
  2  [||||||||                    22.4%]     Load average: 0.39 0.46 0.50
  Mem[|||||||||||||||||||||2085/2505MB]      Uptime: 02:36:33
  Swp[||                     37/1023MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 2322 ubuntu     20   0 1611M  509M 32748 S 24.8 20.3 29:28.99 /usr/lib/firefox1
 1435 ubuntu     10 -10 1611M  509M 32748 S  6.5 20.3  4:32.44 /usr/lib/firefox/
29872 root       20   0 25856  2104  1424 R  5.2  0.1  3:49.00 htop
 2424 ubuntu     20   0 1611M  509M 32748 S  4.6 20.3  2:21.84 /usr/lib/firefox/
 1857 root       20   0  311M  118M  6784 S  3.9  4.7  3:54.87 /usr/bin/X -core
 3454 ubuntu     20   0 3112M  762M  7592 S  2.0 30.4  3:13.50 java -jar target/
  992 ubuntu     20   0 1611M  509M 32748 S  1.3 20.3  1:29.43 /usr/lib/firefox/
 3758 ubuntu     20   0 2590M  253M  7692 S  1.3 10.1  3:43.33 java -Xms200M -Xm
 2603 ubuntu     20   0 1611M  509M 32748 S  1.3 20.3  0:42.10 /usr/lib/firefox/
 3776 ubuntu     20   0 2590M  253M  7692 S  1.3 10.1  0:25.30 java -Xms200M -Xm
 2664 ubuntu     20   0 1611M  509M 32748 S  1.3 20.3  0:08.29 /usr/lib/firefox/
 3794 ubuntu     20   0 2590M  253M  7692 S  1.3 10.1  0:07.94 java -Xms200M -Xm
 3483 ubuntu     20   0 3112M  762M  7592 S  1.3 30.4  0:08.39 java -jar target/
 2336 ubuntu     20   0  409M 14152  8152 S  0.7  0.6  0:33.49 /usr/bin/xfce4-te
 1735 root       10 -10  301M  102M  6692 S  0.7  4.1 10:18.89 ovs-vswitchd unix
 2632 ubuntu     20   0 1611M  509M 32748 S  0.7 20.3  1:03.12 /usr/lib/firefox/
1691 root          F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figure 35 CPU usage on server before attack

Figure 35 shows the CPU usage, 24%, on the web server under normal conditions. CPU cores 1 and 2 have 17% and 22% utilization respectively. Memory usage is at 2085MB of 2505MB and swap is at 37MB of 1023MB. There are 106 tasks and 216 threads but only 1 running process.



	Src Addr	Dst Addr	Proto	Src Prt	Dst Prt
🟦	10.0.0.19	10.0.0.21	6	27715	80
🟥	10.0.0.28	10.0.0.12	6	80	65458
🟧	10.0.0.21	10.0.0.19	6	80	27715
🟩	10.0.0.23	10.0.0.17	6	80	29756
🟪	10.0.0.22	10.0.0.18	1	0	0

Figure 36 Traffic on network before tcp syn attack

The figure above indicates traffic on the network before the tcp syn attack is started. This is normal simulated traffic on the network using the traffic generation script that was discussed earlier. Different colors refer to different hosts on the network i.e.

Blue: Traffic from host 10.0.0.19 to host 10.0.0.21

Red: ICMP traffic from host 10.0.0.28 to host 10.0.0.12

Yellow: ICMP traffic from host 10.0.0.21 to host 10.0.0.19

Green: ICMP traffic from host 10.0.0.23 to host 10.0.0.17

Maroon: ICMP traffic from host 10.0.0.22 to host 10.0.0.18

Total traffic on the network is somewhat just below 300kbps.

At this point the SDN is forwarding all traffic on the network. Flow rules installed on the switch allow the switch to forward all traffic. All hosts can reach each other using any protocol at this point.

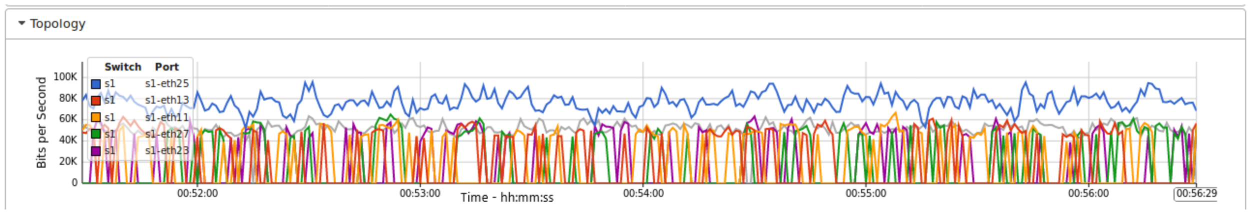


Figure 37 Topology traffic before TCP SYN attack

The figure above shows topology traffic on the network under normal circumstances. This is a representation of traffic as seen on the mininet switch. All hosts connect to this switch. Different colors represent different hosts and ports on the switch e.e.

Blue – switch port 25 for host 10.0.0.25

Red – switch port 13 for host 10.0.0.13

Yellow – switch port 11 for host 10.0.0.11

Green – switch port 27 for host 10.0.0.27

Maroon – switch port 23 for host 10.0.0.23

The graph captures traffic on the whole network from the switch per port.

The SDN controller has installed flow rules that allow all traffic to be forwarded hence all ports on which hosts are connected are able to forward traffic. No host is being blocked.

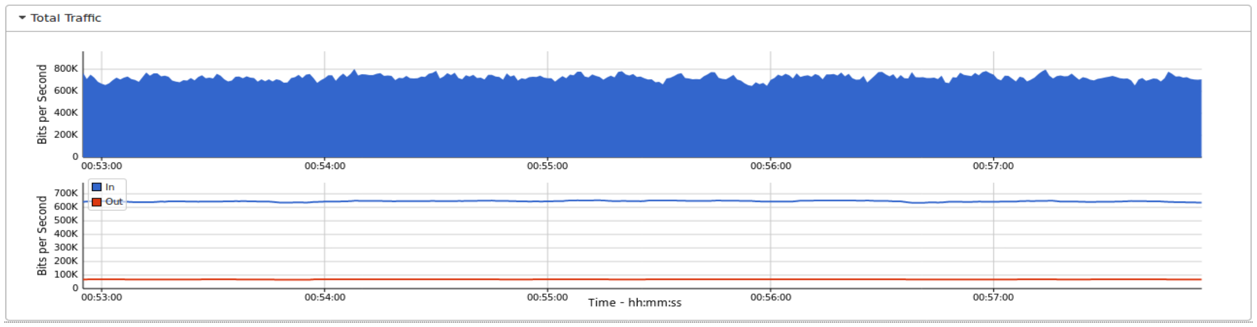


Figure 38 Network traffic

The above image is an indication of total network traffic without the TCP SYN attack traffic. The graph at the top shows all traffic on the virtual network. Total traffic is just below 800kbps. The lower graph indicates incoming traffic using a blue line while outgoing traffic is represented using a red line.

This is the total traffic on the SDN network.

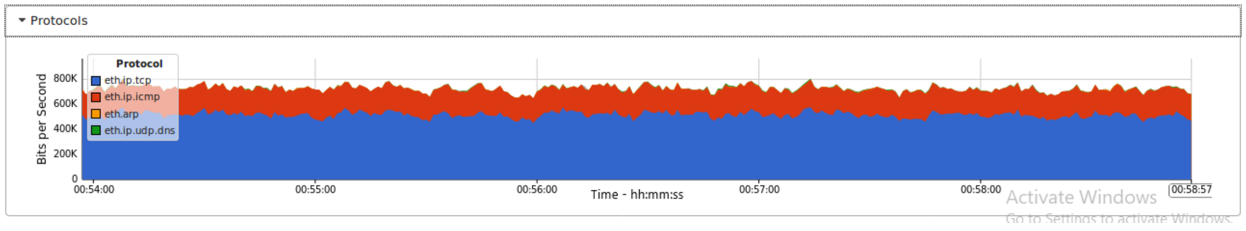


Figure 39 Total protocol traffic

Figure 39 shows total traffic in terms of protocols flowing on the network before simulating the tcp syn attack.

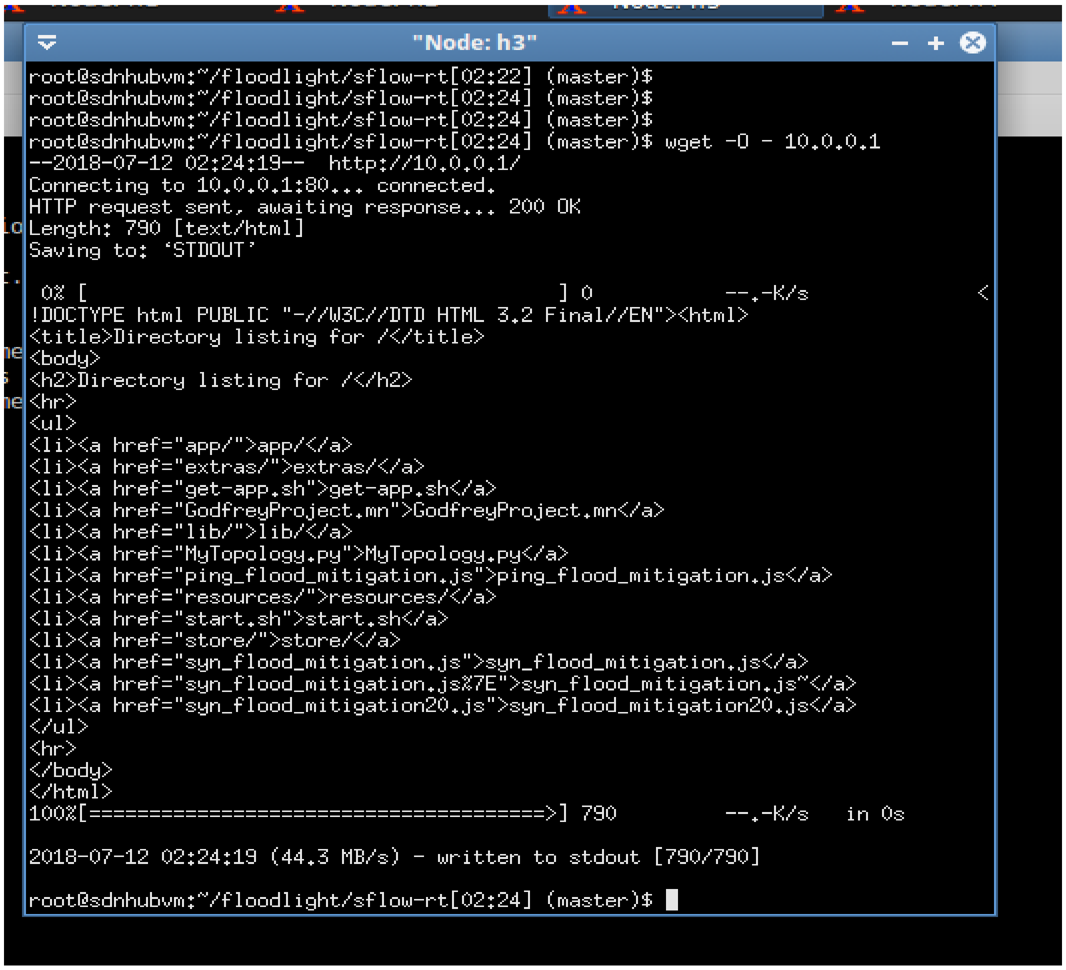
Blue - TCP traffic is represented by the blue section of the graph.

Red - The red indicates IP ICMP traffic and

Yellow shows ARP traffic on the network.

Green shows the UDP DNS traffic.

The SDN is allowing all hosts and all protocols to be forwarded on the network.

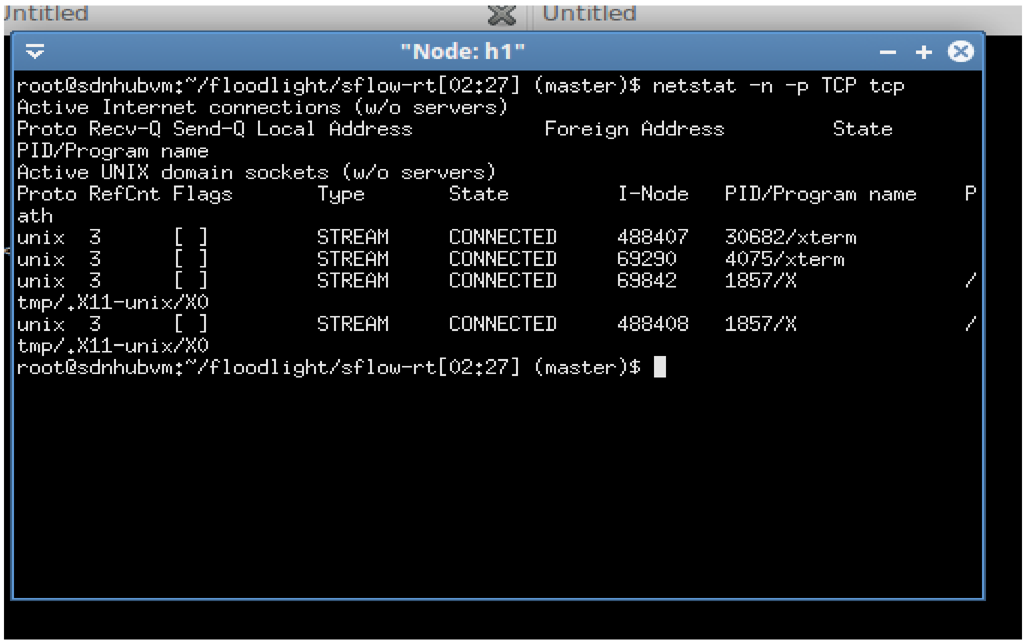


```
root@sdnhubvm:~/floodlight/sflow-rt[02:22] (master)$
root@sdnhubvm:~/floodlight/sflow-rt[02:24] (master)$
root@sdnhubvm:~/floodlight/sflow-rt[02:24] (master)$
root@sdnhubvm:~/floodlight/sflow-rt[02:24] (master)$ wget -O - 10.0.0.1
--2018-07-12 02:24:19--  http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 790 [text/html]
Saving to: ‘STDOUT’

 0% [                                    ] 0           --.-K/s              <
!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href="app/">app/</a>
<li><a href="extras/">extras/</a>
<li><a href="get-app.sh">get-app.sh</a>
<li><a href="GodfreyProject.mn">GodfreyProject.mn</a>
<li><a href="lib/">lib/</a>
<li><a href="MyTopology.py">MyTopology.py</a>
<li><a href="ping_flood_mitigation.js">ping_flood_mitigation.js</a>
<li><a href="resources/">resources/</a>
<li><a href="start.sh">start.sh</a>
<li><a href="store/">store/</a>
<li><a href="syn_flood_mitigation.js">syn_flood_mitigation.js</a>
<li><a href="syn_flood_mitigation.js%7E">syn_flood_mitigation.js~</a>
<li><a href="syn_flood_mitigation20.js">syn_flood_mitigation20.js</a>
</ul>
<hr>
</body>
</html>
100%[===================================>] 790         --.-K/s   in 0s

2018-07-12 02:24:19 (44.3 MB/s) - written to stdout [790/790]

root@sdnhubvm:~/floodlight/sflow-rt[02:24] (master)$
```

Figure 40 Accessing web page

Figure 40 shows that to access a web page from the web server takes less than 1ms. Accessing the web page is done from host h3. This is a simulation of web access from a host, in this case host 3 and the host 1 is configured as the web server hence the IP address 10.0.0.1 in the wget command.

The third last line shows 100% completion of the task and at the end indicates how long it took to download the page – less than 0 seconds!

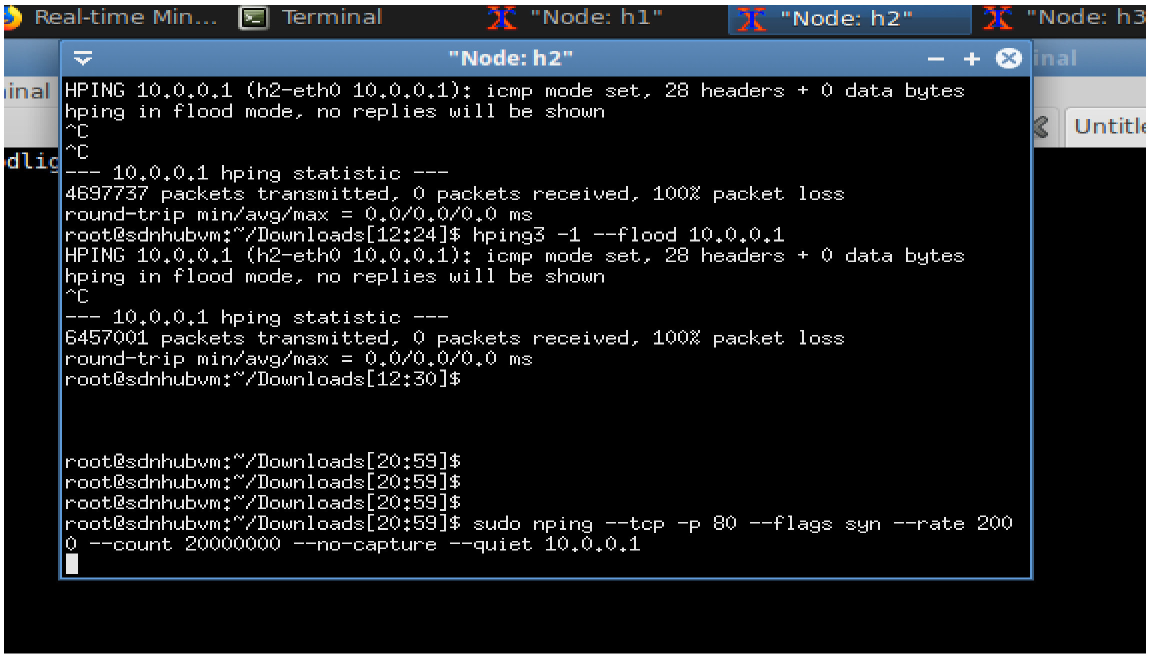


```
root@sdnhubvm:~/floodlight/sflow-rt[02:27] (master)$ netstat -n -p TCP tcp
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node   PID/Program name    P
ath
unix  3      [ ]         STREAM     CONNECTED     488407   30682/xterm
unix  3      [ ]         STREAM     CONNECTED     69290    4075/xterm
unix  3      [ ]         STREAM     CONNECTED     69842    1857/X              /
tmp/.X11-unix/X0
unix  3      [ ]         STREAM     CONNECTED     488408   1857/X              /
tmp/.X11-unix/X0
root@sdnhubvm:~/floodlight/sflow-rt[02:27] (master)$
```

Figure 41 TCP connections on server

Figure 41 shows the server backlog on host 1. All connection attempts have been successful. Since the SDN is not blocking any host, all connection attempts to the server are successful. The web server has received connections and the TCP handshake has succeeded.

4.8.1 Syn attack with nping:



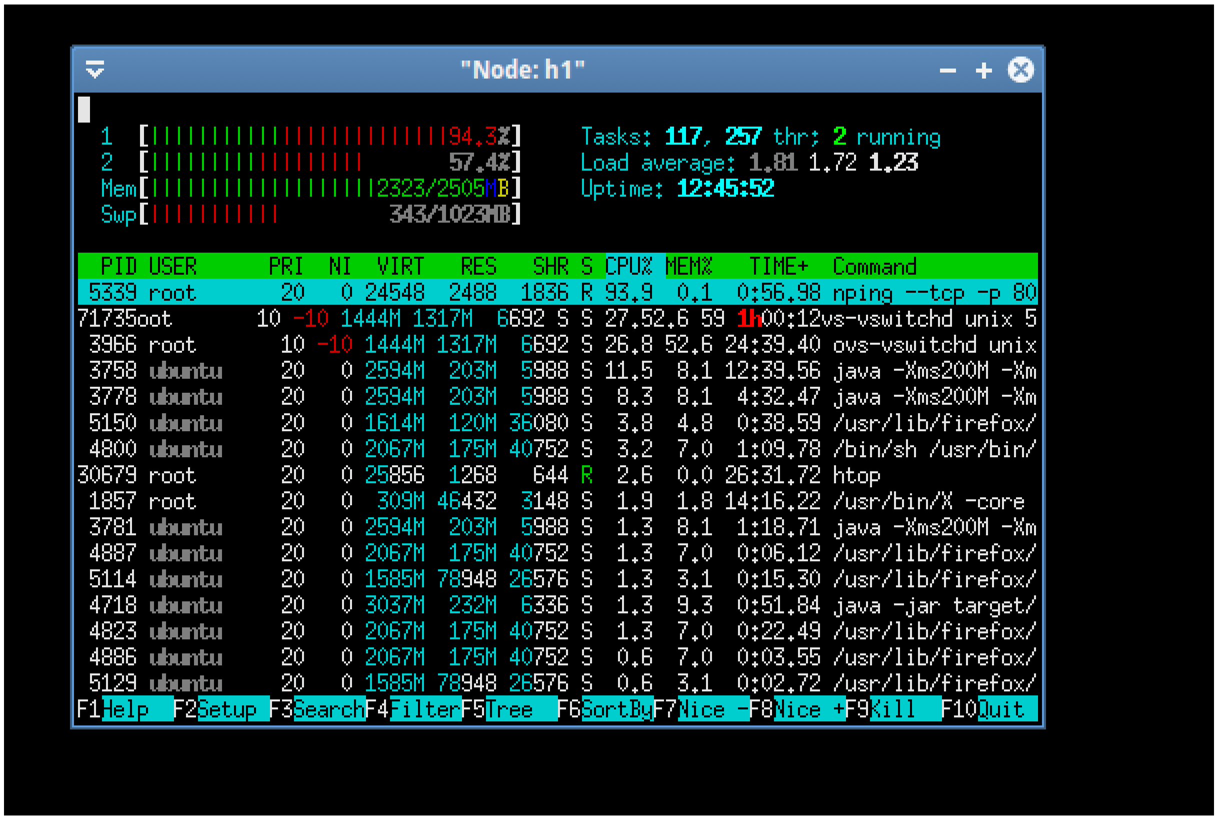
```
HPING 10.0.0.1 (h2-eth0 10.0.0.1): icmp mode set, 28 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
^C
--- 10.0.0.1 hping statistic ---
4697737 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@sdnhubvm:~/Downloads[12:24]$ hping3 -1 --flood 10.0.0.1
HPING 10.0.0.1 (h2-eth0 10.0.0.1): icmp mode set, 28 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 10.0.0.1 hping statistic ---
6457001 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@sdnhubvm:~/Downloads[12:30]$



root@sdnhubvm:~/Downloads[20:59]$
root@sdnhubvm:~/Downloads[20:59]$
root@sdnhubvm:~/Downloads[20:59]$
root@sdnhubvm:~/Downloads[20:59]$ sudo nping --tcp -p 80 --flags syn --rate 200
0 --count 20000000 --no-capture --quiet 10.0.0.1
```

Figure 42 TCP SYN attack with nping

In the figure above, a TCP SYN attack has been initiated from host h2 using the nping command. The attack is aimed at the web server whose IP address is 10.0.0.1.

Source of the attack is host 10.0.0.2.

4.8.2 CPU usage on server:



```
  1  [||||||||||||||||||||||||||||||94.3%]     Tasks: 117, 257 thr; 2 running
  2  [||||||||||||||||||         57.4%]     Load average: 1.81 1.72 1.23
  Mem[||||||||||||||||||||2323/2505MB]     Uptime: 12:45:52
  Swp[|||||||||||          343/1023MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 5339 root       20   0 24548  2488  1836 R 93.9  0.1  0:56.98 nping --tcp -p 80
71735oot        10 -10 1444M 1317M  6692 S S 27.52.6 59 1h00:12vs-vswitchd unix 5
 3966 root       10 -10 1444M 1317M  6692 S 26.8 52.6 24:39.40 ovs-vswitchd unix
 3758 ubuntu     20   0 2594M  203M  5988 S 11.5  8.1 12:39.56 java -Xms200M -Xm
 3778 ubuntu     20   0 2594M  203M  5988 S  8.3  8.1  4:32.47 java -Xms200M -Xm
 5150 ubuntu     20   0 1614M  120M 36080 S  3.8  4.8  0:38.59 /usr/lib/firefox/
 4800 ubuntu     20   0 2067M  175M 40752 S  3.2  7.0  1:09.78 /bin/sh /usr/bin/
30679 root       20   0 25856  1268   644 R  2.6  0.0 26:31.72 htop
 1857 root       20   0  309M 46432  3148 S  1.9  1.8 14:16.22 /usr/bin/X -core
 3781 ubuntu     20   0 2594M  203M  5988 S  1.3  8.1  1:18.71 java -Xms200M -Xm
 4887 ubuntu     20   0 2067M  175M 40752 S  1.3  7.0  0:06.12 /usr/lib/firefox/
 5114 ubuntu     20   0 1585M 78948 26576 S  1.3  3.1  0:15.30 /usr/lib/firefox/
 4718 ubuntu     20   0 3037M  232M  6336 S  1.3  9.3  0:51.84 java -jar target/
 4823 ubuntu     20   0 2067M  175M 40752 S  1.3  7.0  0:22.49 /usr/lib/firefox/
 4886 ubuntu     20   0 2067M  175M 40752 S  0.6  7.0  0:03.55 /usr/lib/firefox/
 5129 ubuntu     20   0 1585M 78948 26576 S  0.6  3.1  0:02.72 /usr/lib/firefox/
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figure 43 CPU usage on server

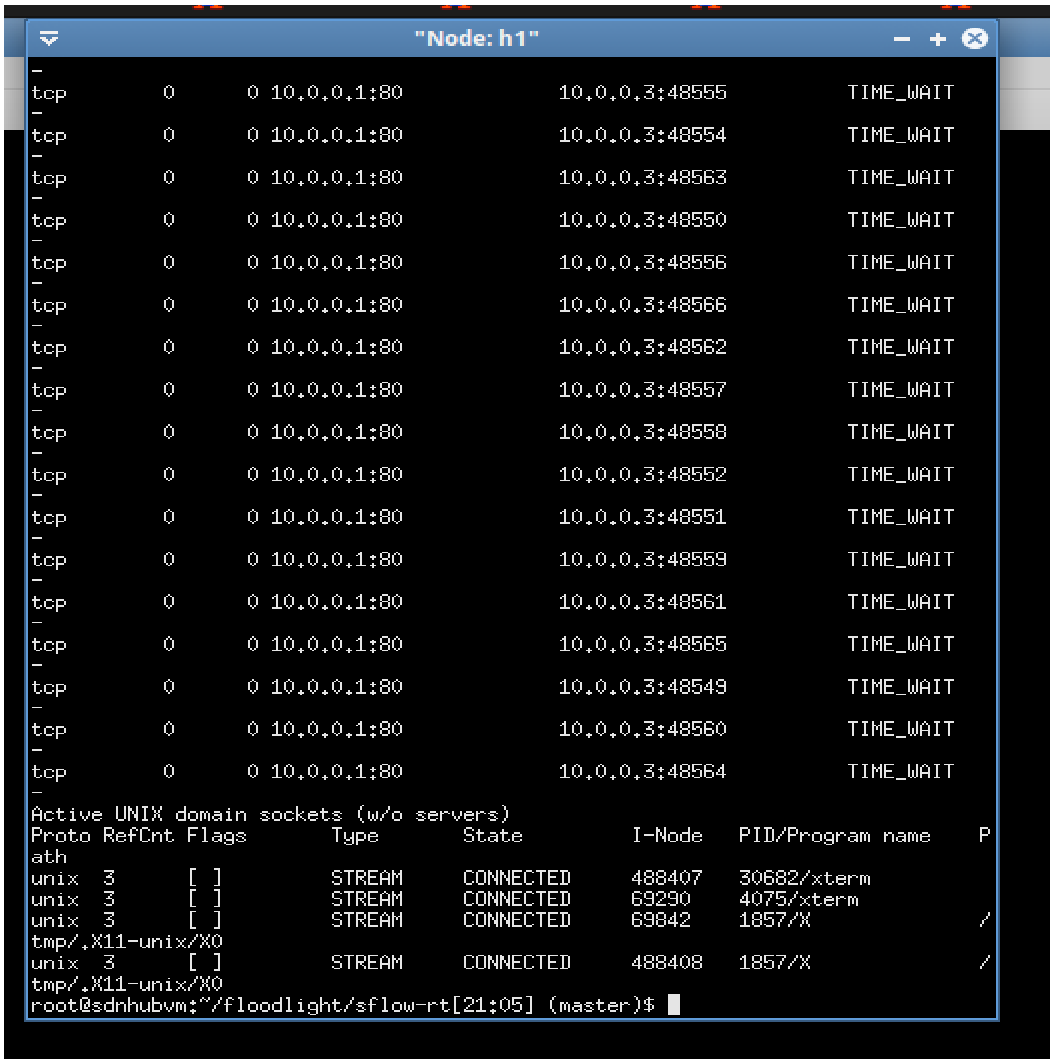
Figure 43 shows that CPU usage shoots up from 24% to 93.9%. The web server's CPU is high due to the tcp syn attack initiated above. Both CPU cores 1 and 2 have also had their usage go high up to 94% and 57% respectively.

Memory usage is at 2323MB out of 2505MB. Swap usage is also at 343MB out of 1023MB.

117 tasks and 257 threads exist on the system and only 2 were running.

At this point the SDN is vulnerable because the controller has no intelligence to identify malicious traffic. Hence, all traffic is forwarded to the server leading to exhaustion of resources such as CPU. Eventually the services will be inaccessible because the server is overwhelmed and cannot process any requests.

4.8.3 TCP Connections on Server:



```
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48555          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48554          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48563          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48550          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48556          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48566          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48562          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48557          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48558          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48552          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48551          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48559          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48561          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48565          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48549          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48560          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48564          TIME_WAIT
-
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node   PID/Program name    P
ath
unix  3      [ ]         STREAM     CONNECTED     488407   30682/xterm
unix  3      [ ]         STREAM     CONNECTED     69290    4075/xterm
unix  3      [ ]         STREAM     CONNECTED     69842    1857/X              /
tmp/.X11-unix/X0
unix  3      [ ]         STREAM     CONNECTED     488408   1857/X              /
tmp/.X11-unix/X0
root@sdnhubvm:~/floodlight/sflow-rt[21:05] (master)$
```

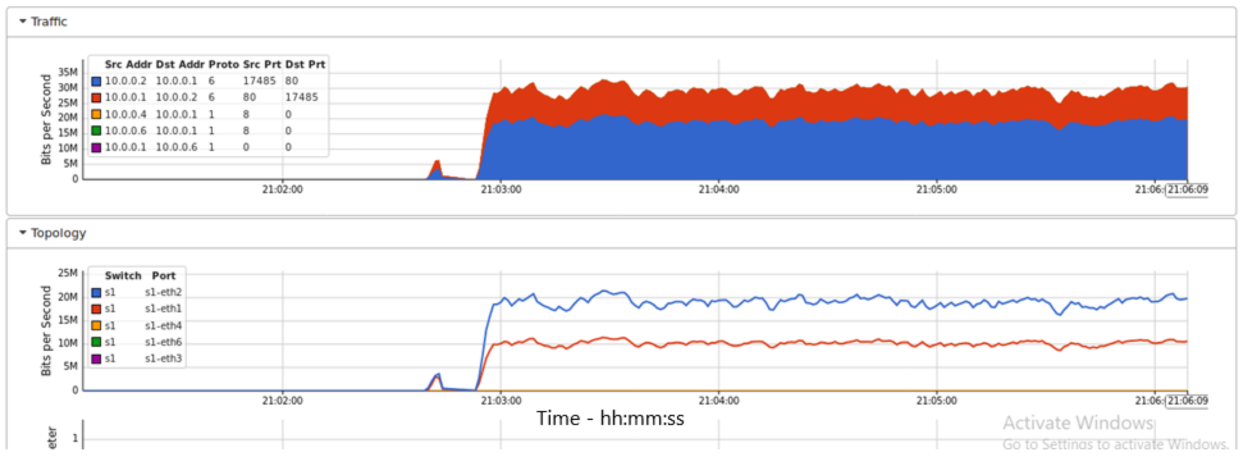
Figure 44 TCP connections on server

Figure 44 shows the server's back log. Most of the connections are in TIME_WAIT. The attack has made the server unavailable thus it cannot process the connection requests.

Connections to the server are stuck in TIME_WAIT. The server is overwhelmed hence it cannot process connection requests. The attack has succeeded in making web server services unavailable to legitimate users.

This was possible because at this stage, the SDN has no way of detecting and stopping the TCP SYN attack.

4.8.4 Graph on network:



	Src Addr	Dst Addr	Proto	Src Prt	Dst Prt
🟦	10.0.0.2	10.0.0.1	6	17485	80
🟥	10.0.0.1	10.0.0.2	6	80	17485
🟧	10.0.0.4	10.0.0.1	1	8	0
🟩	10.0.0.6	10.0.0.1	1	8	0
🟪	10.0.0.1	10.0.0.6	1	0	0

	Switch	Port
🟦	s1	s1-eth2
🟥	s1	s1-eth1
🟧	s1	s1-eth4
🟩	s1	s1-eth6
🟪	s1	s1-eth3

Figure 45 Graph showing an attack on the server

Figure 45 shows total traffic and topology traffic on the network after the tcp syn attack is initiated. Traffic shoots up indicating a ddos attack on the network. The top graph shows traffic sources and destinations using different colors i.e.

Blue – TCP traffic from 10.0.0.2(attacker) to 10.0.0.1(web server).

Red – TCP traffic from 10.0.0.1, web server, back to 10.0.0.2, which is the attacking machine.

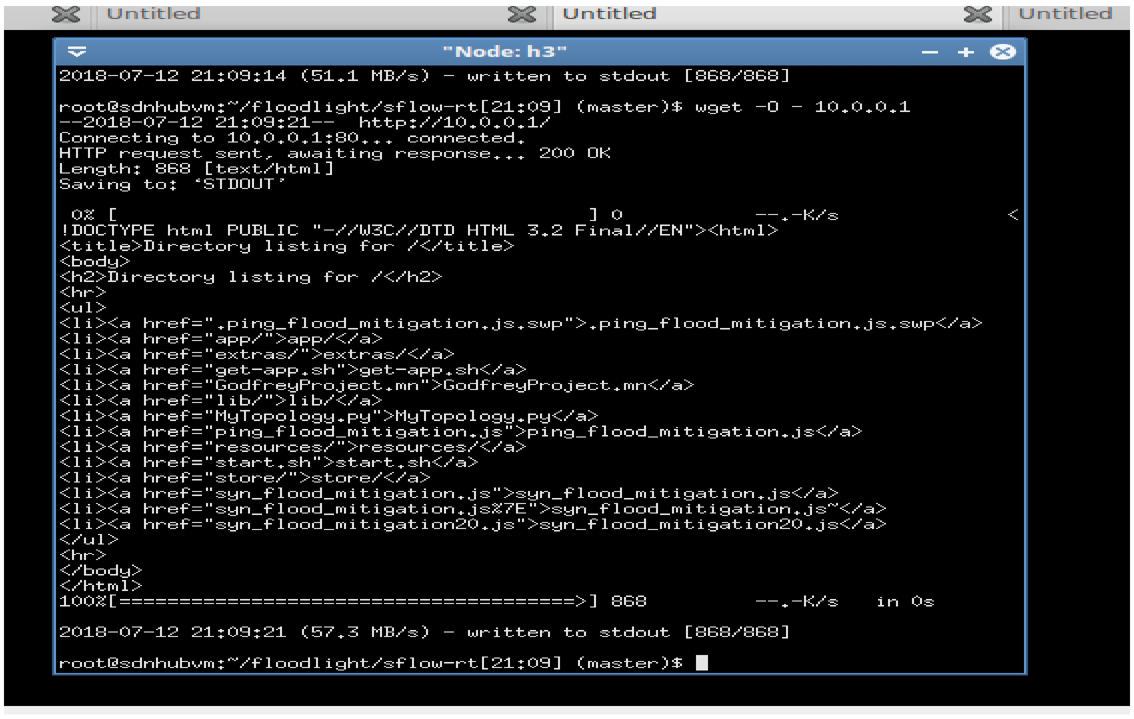
Yellow – TCP traffic from 10.0.0.4 to 10.0.0.1

Green – TCP traffic from 10.0.0.6 to 10.0.0.1

Purple – TCP traffic from 10.0.0.22 to 10.0.0.6

At this stage, the SDN does not have the application running that can detect DDoS and stop it. Hence, the traffic is allowed through the network and ends up overwhelming the web server.

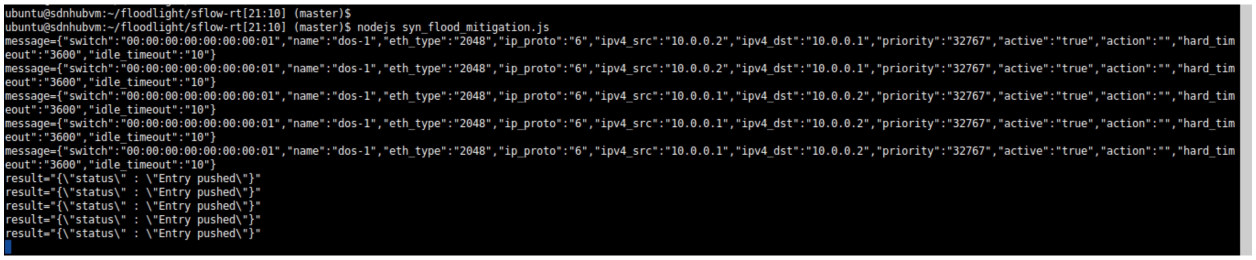
4.8.5 Web server Access:



```
2018-07-12 21:09:14 (51.1 MB/s) - written to stdout [868/868]

root@sdnhubvm:~/floodlight/sflow-rt[21:09] (master)$ wget -O - 10.0.0.1
--2018-07-12 21:09:21--  http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 868 [text/html]
Saving to: 'STDOUT'

 0% [                                       ] 0           --.-K/s              <
!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href=".ping_flood_mitigation.js.swp">.ping_flood_mitigation.js.swp</a>
<li><a href="app/">app/</a>
<li><a href="extras/">extras/</a>
<li><a href="get-app.sh">get-app.sh</a>
<li><a href="GodfreyProject.mn">GodfreyProject.mn</a>
<li><a href="lib/">lib/</a>
<li><a href="MyTopology.py">MyTopology.py</a>
<li><a href="ping_flood_mitigation.js">ping_flood_mitigation.js</a>
<li><a href="resources/">resources/</a>
<li><a href="start.sh">start.sh</a>
<li><a href="store/">store/</a>
<li><a href="syn_flood_mitigation.js">syn_flood_mitigation.js</a>
<li><a href="syn_flood_mitigation.js%7E">syn_flood_mitigation.js~</a>
<li><a href="syn_flood_mitigation20.js">syn_flood_mitigation20.js</a>
</ul>
<hr>
</body>
</html>
100%[======================================>] 868         --.-K/s   in 0s

2018-07-12 21:09:21 (57.3 MB/s) - written to stdout [868/868]

root@sdnhubvm:~/floodlight/sflow-rt[21:09] (master)$ 
```

Figure 46 Web access during attack

4.8.6 SYN Mitigation



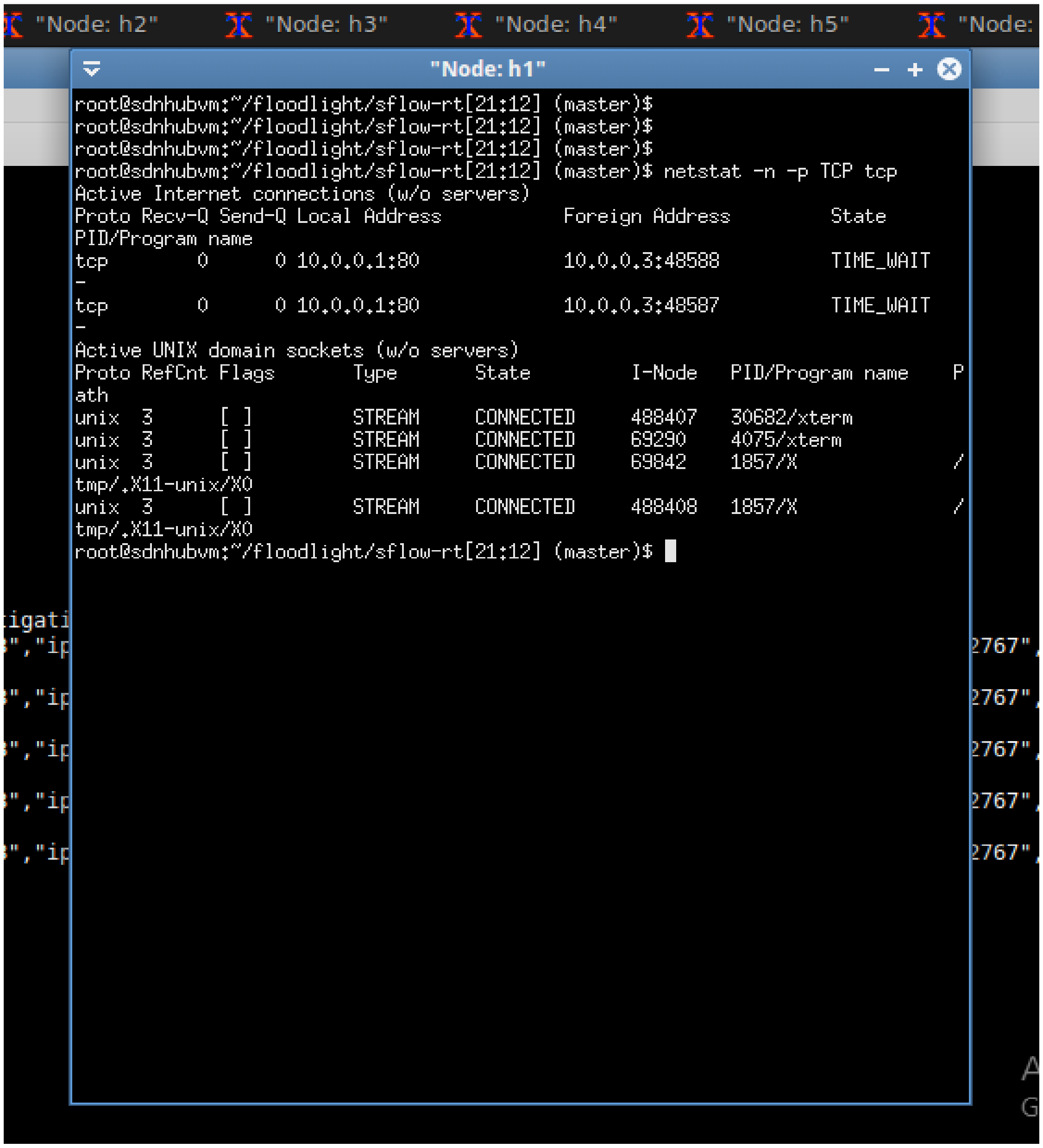
```
ubuntu@sdnhubvm:~/floodlight/sflow-rt[21:10] (master)$
ubuntu@sdnhubvm:~/floodlight/sflow-rt[21:10] (master)$ nodejs syn_flood_mitigation.js
message={"switch":"00:00:00:00:00:00:00:01","name":"dos-1","eth_type":"2048","ip_proto":"6","ipv4_src":"10.0.0.2","ipv4_dst":"10.0.0.1","priority":"32767","active":"true","action":"","hard_tim
eout":"3600","idle_timeout":"10"}
message={"switch":"00:00:00:00:00:00:00:01","name":"dos-1","eth_type":"2048","ip_proto":"6","ipv4_src":"10.0.0.2","ipv4_dst":"10.0.0.1","priority":"32767","active":"true","action":"","hard_tim
eout":"3600","idle_timeout":"10"}
message={"switch":"00:00:00:00:00:00:00:01","name":"dos-1","eth_type":"2048","ip_proto":"6","ipv4_src":"10.0.0.1","ipv4_dst":"10.0.0.2","priority":"32767","active":"true","action":"","hard_tim
eout":"3600","idle_timeout":"10"}
message={"switch":"00:00:00:00:00:00:00:01","name":"dos-1","eth_type":"2048","ip_proto":"6","ipv4_src":"10.0.0.1","ipv4_dst":"10.0.0.2","priority":"32767","active":"true","action":"","hard_tim
eout":"3600","idle_timeout":"10"}
message={"switch":"00:00:00:00:00:00:00:01","name":"dos-1","eth_type":"2048","ip_proto":"6","ipv4_src":"10.0.0.1","ipv4_dst":"10.0.0.2","priority":"32767","active":"true","action":"","hard_tim
eout":"3600","idle_timeout":"10"}
result="{\"status\" : \"Entry pushed\"}"
result="{\"status\" : \"Entry pushed\"}"
result="{\"status\" : \"Entry pushed\"}"
result="{\"status\" : \"Entry pushed\"}"
result="{\"status\" : \"Entry pushed\"}"
```

Figure 47 Mitigation of attack

The figure below shows the mitigation script being executed. The script is an application running on top of the controller, giving it the intelligence to detect malicious flows. It then installs flows on the switches to counter the malicious traffic. The switch is instructed to drop all traffic emanating from the malicious source.

The SDN controller now has the intelligence to identify TCP SYN attack. It was able to determine that the source of the attack was from the IP address 10.0.0.2. Upon this discovery, it installed appropriated flow rules, blocking any traffic from 10.0.0.2 destined to 10.0.0.1, onto the switch. From that instance, any traffic matching that flow rule is dropped thus stopping the TCP SYN attack.

4.8.6.1 Server Connections:



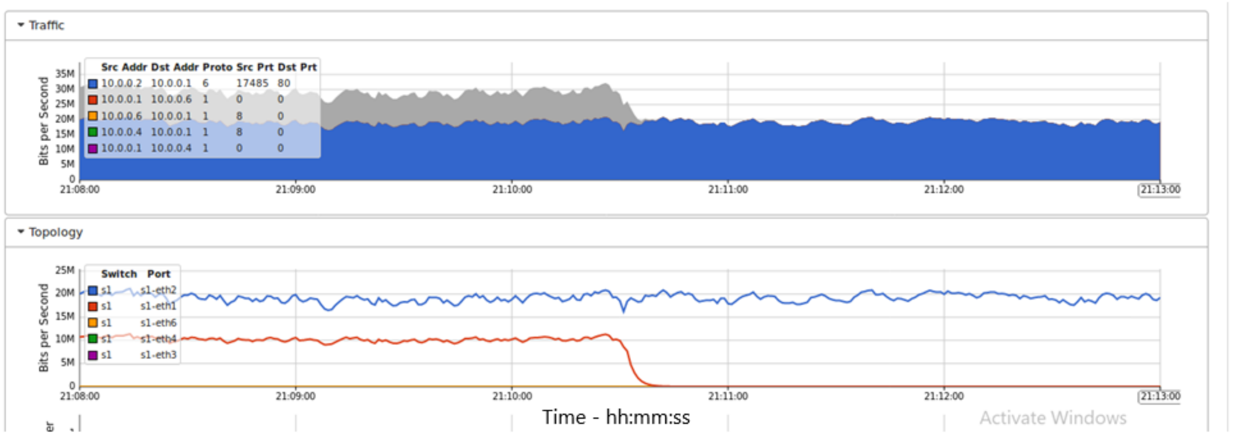
```
root@sdnhubvm:~/floodlight/sflow-rt[21:12] (master)$
root@sdnhubvm:~/floodlight/sflow-rt[21:12] (master)$
root@sdnhubvm:~/floodlight/sflow-rt[21:12] (master)$
root@sdnhubvm:~/floodlight/sflow-rt[21:12] (master)$ netstat -n -p TCP tcp
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 10.0.0.1:80             10.0.0.3:48588          TIME_WAIT
-
tcp        0      0 10.0.0.1:80             10.0.0.3:48587          TIME_WAIT
-
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node   PID/Program name    P
ath
unix  3      [ ]         STREAM     CONNECTED     488407   30682/xterm
unix  3      [ ]         STREAM     CONNECTED     69290    4075/xterm
unix  3      [ ]         STREAM     CONNECTED     69842    1857/X              /
tmp/.X11-unix/X0
unix  3      [ ]         STREAM     CONNECTED     488408   1857/X              /
tmp/.X11-unix/X0
root@sdnhubvm:~/floodlight/sflow-rt[21:12] (master)$
```

Figure 48 Connections on server

In the above image, the server backlog has reduced and more connections are successful.

The SDN controller having stopped the attack from 10.0.0.2, reduces load on the server, enabling it to allow more connections.

4.8.7 Usage on network



	Src Addr	Dst Addr	Proto	Src Prt	Dst Prt
🟦	10.0.0.2	10.0.0.1	6	17485	80
🟥	10.0.0.1	10.0.0.6	1	0	0
🟧	10.0.0.6	10.0.0.1	1	8	0
🟩	10.0.0.4	10.0.0.1	1	8	0
🟪	10.0.0.1	10.0.0.4	1	0	0

	Switch	Port
🟦	s1	s1-eth2
🟥	s1	s1-eth1
🟧	s1	s1-eth6
🟩	s1	s1-eth4
🟪	s1	s1-eth3

Figure 49 Traffic drop after mitigation

Figure 49 shows DDoS traffic being dropped by the switch. The traffic from the attacker continues to reach the switch. However, this traffic is not transmitted to the intended destination as the flow entry has been deleted to protect the web server from the attack.

The graph at the top shows mostly traffic from the attacker, 10.0.0.1 destined for the web server 10.0.0.1, in blue.

The graph does not show traffic from the web server, 10.0.0.1 back to the attacker 10.0.0.2.

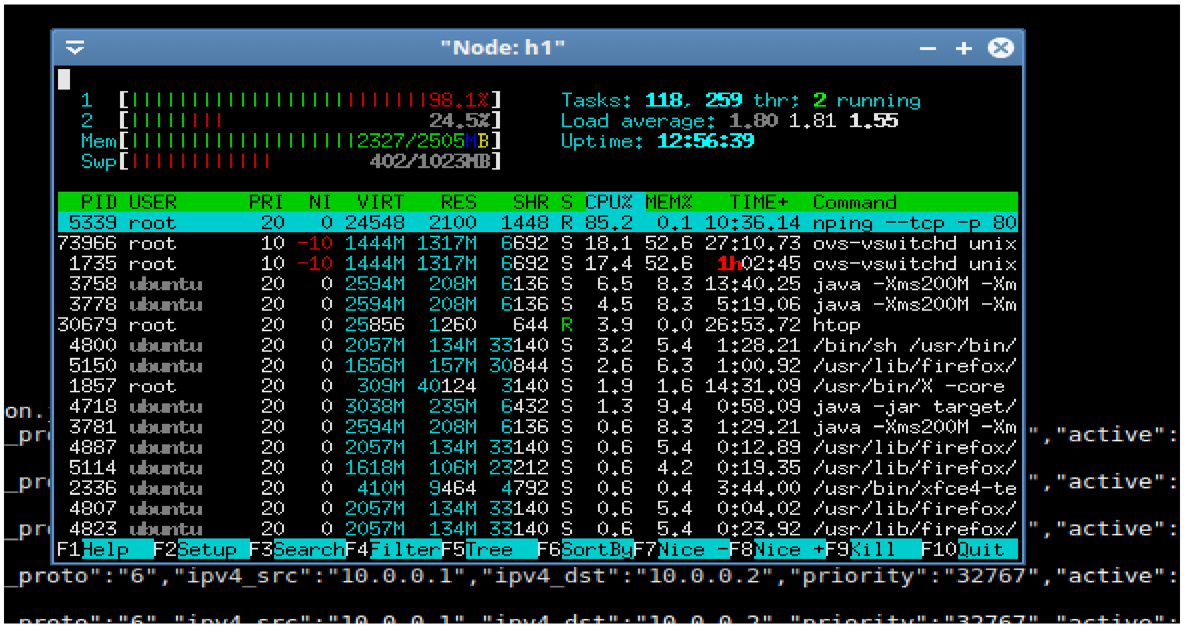
The controller has dropped the flow allowing this kind of traffic after it recognized it as malicious traffic.

When traffic from 10.0.0.2 reaches the switch, it is not forwarded to the server thus protecting the server from the DDoS attack. The traffic is dropped as per the new flow rule installed by the controller onto the mininet switch.

The other traffic types are marginal hence not shown in color on the graph.

The SDN controller detected the TCP SYN attack. It then created flow rules instructing the switch to drop any traffic from 10.0.0.2 to 10.0.0.1. So traffic from 10.0.0.2 continues to arrive on the switch but is not forwarded to 10.0.0.1, hence on the graph, it continues to show that traffic on the port that 10.0.0.2 is connected on still has high traffic. However, this traffic is not forwarded to 10.0.0.1, which is the web server. So no response or traffic is seen to arrive on the port that 10.0.0.1 is connected on, the red line. So there is a sharp drop on the line graph.

The SDN has then nullified the DDoS, in the process protecting the web server from malicious traffic.



```
  1  [|||||||||||||||||||||||||||98.1%]     Tasks: 118, 259 thr: 2 running
  2  [||||||||                   24.5%]     Load average: 1.80 1.81 1.55
  Mem[||||||||||||||||||||||2327/2505MB]    Uptime: 12:56:39
  Swp[|||||||||||            402/1023MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 5339 root       20   0 24548  2100  1448 R 85.2  0.1 10:36.14 nping --tcp -p 80
73966 root       10 -10 1444M 1317M  6692 S 18.1 52.6 27:10.73 ovs-vswitchd unix
 1735 root       10 -10 1444M 1317M  6692 S 17.4 52.6  1h02:45 ovs-vswitchd unix
 3758 ubuntu     20   0 2594M  208M  6136 S  6.5  8.3 13:40.25 java -Xms200M -Xm
 3778 ubuntu     20   0 2594M  208M  6136 S  4.5  8.3  5:19.06 java -Xms200M -Xm
30679 root       20   0 25856  1260   644 R  3.9  0.0 26:53.72 htop
 4800 ubuntu     20   0 2057M  134M 33140 S  3.2  5.4  1:28.21 /bin/sh /usr/bin/
 5150 ubuntu     20   0 1656M  157M 30844 S  2.6  6.3  1:00.92 /usr/lib/firefox/
 1857 root       20   0  309M 40124  3140 S  1.9  1.6 14:31.09 /usr/bin/X -core
 4718 ubuntu     20   0 3038M  235M  6432 S  1.3  9.4  0:58.09 java -jar target/
 3781 ubuntu     20   0 2594M  208M  6136 S  0.6  8.3  1:29.21 java -Xms200M -Xm
 4887 ubuntu     20   0 2057M  134M 33140 S  0.6  5.4  0:12.89 /usr/lib/firefox/
 5114 ubuntu     20   0 1618M  106M 23212 S  0.6  4.2  0:19.35 /usr/lib/firefox/
 2336 ubuntu     20   0  410M  9464  4792 S  0.6  0.4  3:44.00 /usr/bin/xfce4-te
 4807 ubuntu     20   0 2057M  134M 33140 S  0.6  5.4  0:04.02 /usr/lib/firefox/
 4823 ubuntu     20   0 2057M  134M 33140 S  0.6  5.4  0:23.92 /usr/lib/firefox/
F1Help  F2Setup  F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
_proto":"6","ipv4_src":"10.0.0.1","ipv4_dst":"10.0.0.2","priority":"32767","active":
```

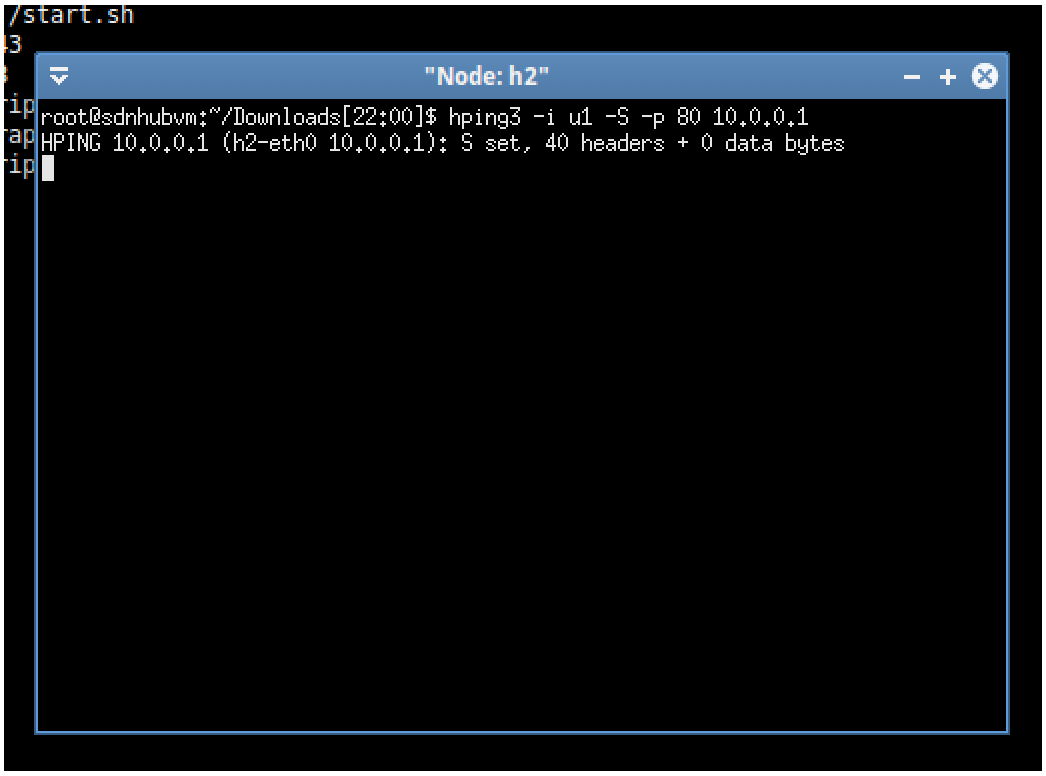
Figure 50 CPU usage

CPU usage is seen to be reducing to normal levels in the above image. CPU core 2 has had its usage drop while that of CPU core is still reducing. Overall CPU usage has dropped to 85%. Tasks and threads remain almost the same.

Memory and swap usage are not affected much.

The SDN controller identified malicious flows i.e. traffic from 10.0.0.2 to 10.0.0.1. It then installed flows instructing the switch to drop this type of flows thus protecting the server from the TCP SYN attack. Any traffic from 10.0.0.2 to 10.0.0.1 is dropped. Hence resources on the server are not exhausted e.g. the CPU usage.

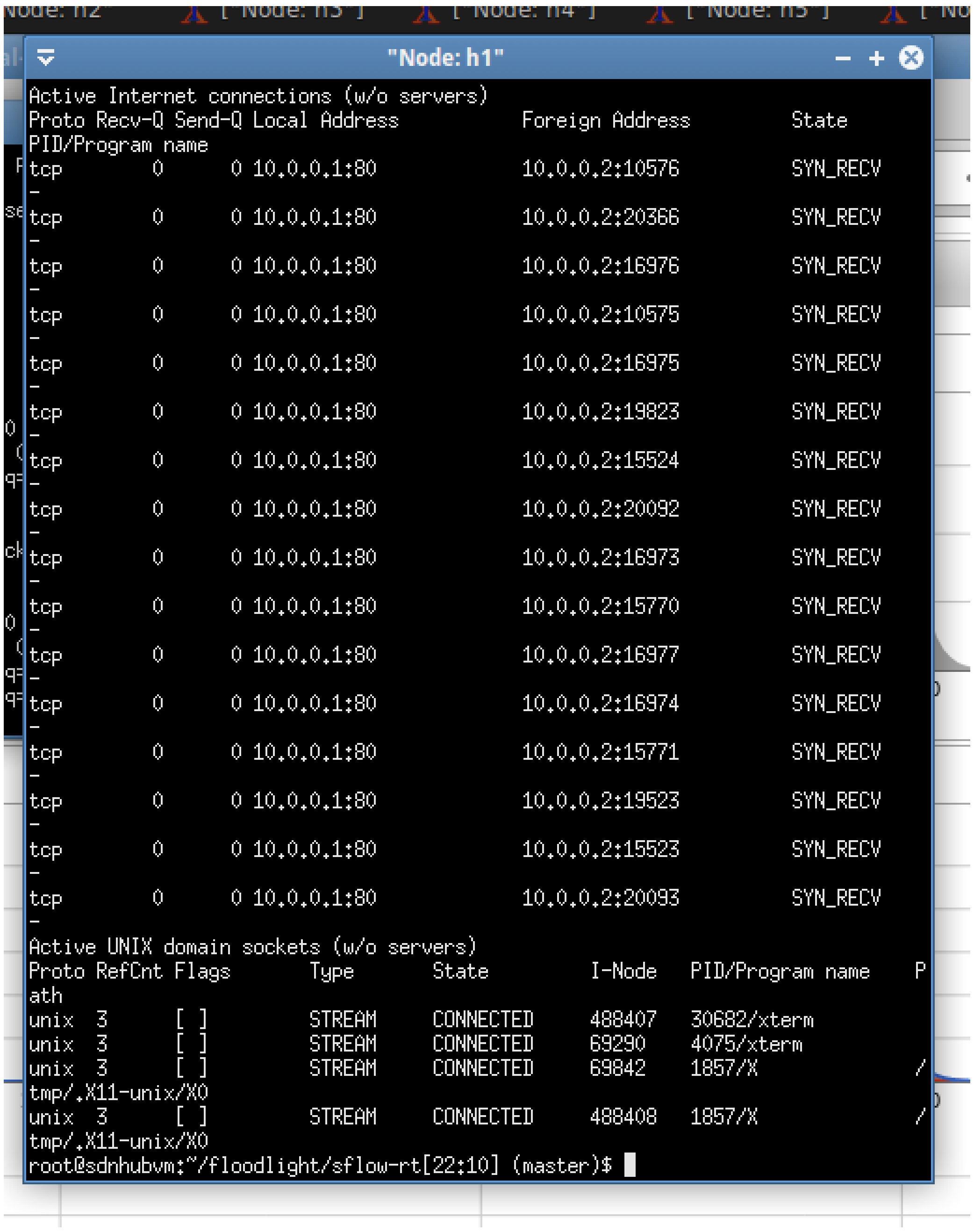
4.8.8 SYN Attack with hping3



```
root@sdnhubvm:~/Downloads[22:00]$ hping3 -i u1 -S -p 80 10.0.0.1
HPING 10.0.0.1 (h2-eth0 10.0.0.1): S set, 40 headers + 0 data bytes
```

Figure 51 TCP SYN attack with hping3

In figure 51, a tcp syn attack is implemented using hping3 utility.

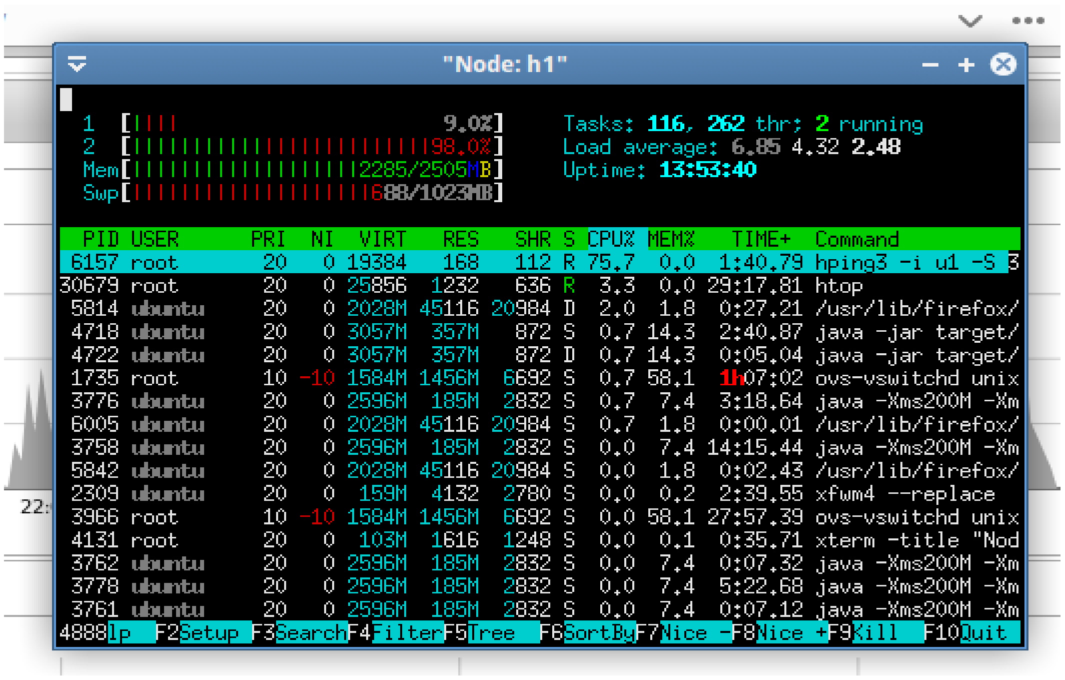


```
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 10.0.0.1:80             10.0.0.2:10576          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:20366          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:16976          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:10575          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:16975          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:19823          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:15524          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:20092          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:16973          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:15770          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:16977          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:16974          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:15771          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:19523          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:15523          SYN_RECV
-
tcp        0      0 10.0.0.1:80             10.0.0.2:20093          SYN_RECV
-
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node   PID/Program name    P
ath
unix  3      [ ]         STREAM     CONNECTED     488407   30682/xterm
unix  3      [ ]         STREAM     CONNECTED     69290    4075/xterm
unix  3      [ ]         STREAM     CONNECTED     69842    1857/X              /
tmp/.X11-unix/X0
unix  3      [ ]         STREAM     CONNECTED     488408   1857/X              /
tmp/.X11-unix/X0
root@sdnhubvm:~/floodlight/sflow-rt[22:10] (master)$
```

Figure 52 server connections

The above image shows the server backlog with most of the connections showing how the server has received numerous SYN packets. The connections are not completing to the CONNECTED state.

The SDN controller has not application to help it detect and stop DDoS attacks at this point. All traffic to and from all hosts is allowed on the switch. The attack originated from 10.0.0.2 is allowed to continue unabated as the controller cannot tell that it’s a DDoS attack. Hence all traffic reaches the web server, exhausts all the resources eventually making the server unresponsive to all requests.



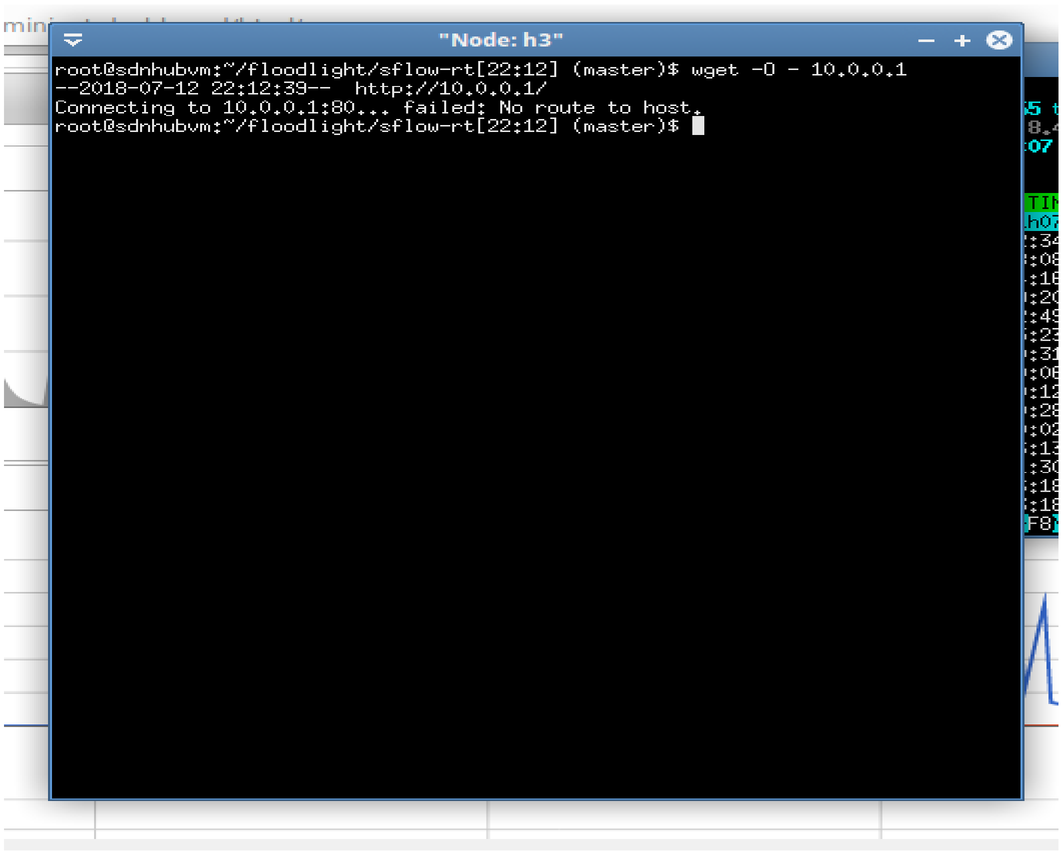
```
  1  [||||                     9.0%]     Tasks: 116, 262 thr; 2 running
  2  [||||||||||||||||||||||||98.0%]     Load average: 6.85 4.32 2.48
  Mem[||||||||||||||||||||2285/2505MB]   Uptime: 13:53:40
  Swp[||||||||||||||||||||688/1023MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 6157 root       20   0 19384   168   112 R 75.7  0.0  1:40.79 hping3 -i u1 -S 3
30679 root       20   0 25856  1232   636 R  3.3  0.0 29:17.81 htop
 5814 ubuntu     20   0 2028M 45116 20984 D  2.0  1.8  0:27.21 /usr/lib/firefox/
 4718 ubuntu     20   0 3057M  357M   872 S  0.7 14.3  2:40.87 java -jar target/
 4722 ubuntu     20   0 3057M  357M   872 D  0.7 14.3  0:05.04 java -jar target/
 1735 root       10 -10 1584M 1456M  6692 S  0.7 58.1  1h07:02 ovs-vswitchd unix
 3776 ubuntu     20   0 2596M  185M  2832 S  0.7  7.4  3:18.64 java -Xms200M -Xm
 6005 ubuntu     20   0 2028M 45116 20984 S  0.7  1.8  0:00.01 /usr/lib/firefox/
 3758 ubuntu     20   0 2596M  185M  2832 S  0.0  7.4 14:15.44 java -Xms200M -Xm
 5842 ubuntu     20   0 2028M 45116 20984 S  0.0  1.8  0:02.43 /usr/lib/firefox/
 2309 ubuntu     20   0  159M  4132  2780 S  0.0  0.2  2:39.55 xfwm4 --replace
 3966 root       10 -10 1584M 1456M  6692 S  0.0 58.1 27:57.39 ovs-vswitchd unix
 4131 root       20   0  103M  1616  1248 S  0.0  0.1  0:35.71 xterm -title "Nod
 3762 ubuntu     20   0 2596M  185M  2832 S  0.0  7.4  0:07.32 java -Xms200M -Xm
 3778 ubuntu     20   0 2596M  185M  2832 S  0.0  7.4  5:22.68 java -Xms200M -Xm
 3761 ubuntu     20   0 2596M  185M  2832 S  0.0  7.4  0:07.12 java -Xms200M -Xm
4888lp   F2Setup  F3SearchF4FilterF5Tree   F6SortByF7Nice -F8Nice +F9Kill   F10Quit
```

Figure 53 CPU usage on server

CPU usage is seen to be increasing in the image above due to the tcp syn attack launched above. Usage on the two CPU cores is also increasing. Overall CPU usage is at 75%.

Since the SDN controller cannot tell that a DDoS attack is in progress, it allowed all traffic from 10.0.0.2 to reach the web server until it was eventually overwhelmed and unable to process any requests, both legitimate and illegitimate.

The DDoS is successfully executed from 10.0.0.2 making web services unavailable from 10.0.0.1. Once the CPU usage is so high, the server is incapable of processing any requests further.

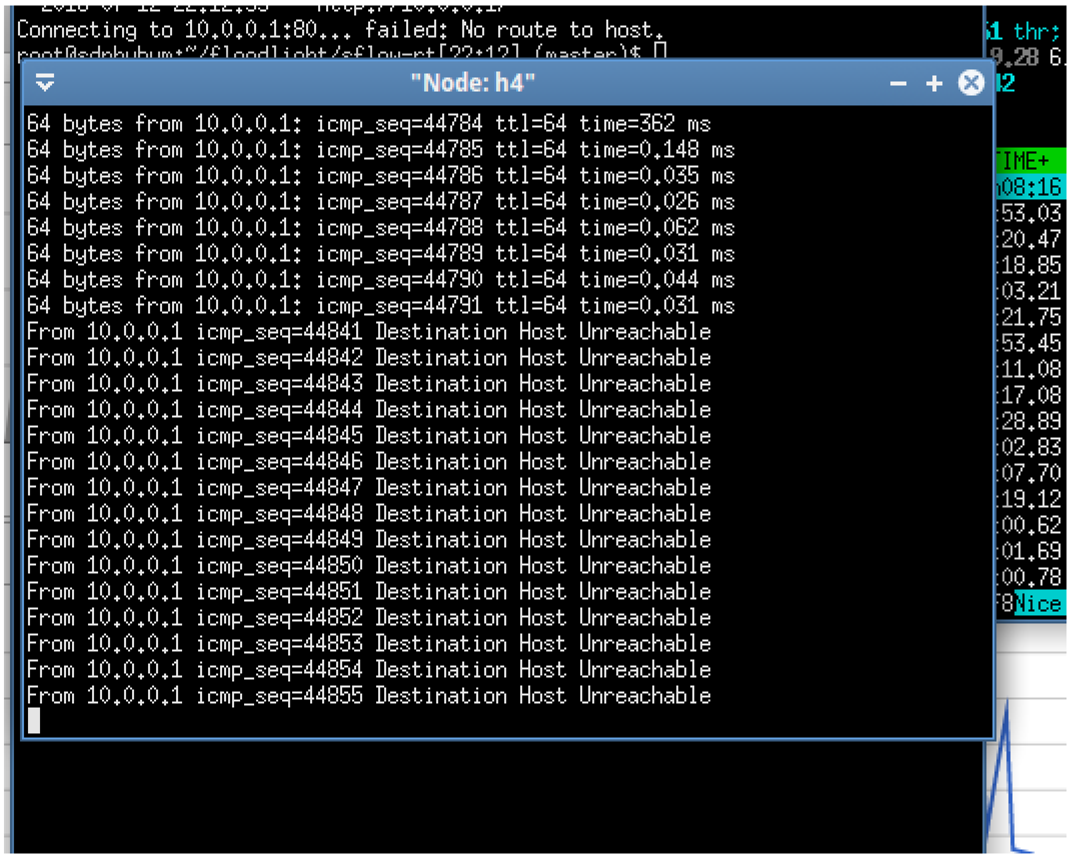


```
root@sdnhubvm:~/floodlight/sflow-rt[22:12] (master)$ wget -O - 10.0.0.1
--2018-07-12 22:12:39--  http://10.0.0.1/
Connecting to 10.0.0.1:80... failed: No route to host.
root@sdnhubvm:~/floodlight/sflow-rt[22:12] (master)$
```

Figure 54 Web server access

Attempts to access the web server are unsuccessful due to the ddos attack initiated above. The web server has been overwhelmed and cannot respond at this instance. The web page cannot be accessed.

At this moment, the SDN controller has no intelligence to detect this attack hence the web server is left exposed. The SDN network forwards all traffic leading to exhaustion of resources on the server and making it unable to process any further requests.



```
Connecting to 10.0.0.1:80... failed: No route to host.
```

```
64 bytes from 10.0.0.1: icmp_seq=44784 ttl=64 time=362 ms
64 bytes from 10.0.0.1: icmp_seq=44785 ttl=64 time=0.148 ms
64 bytes from 10.0.0.1: icmp_seq=44786 ttl=64 time=0.035 ms
64 bytes from 10.0.0.1: icmp_seq=44787 ttl=64 time=0.026 ms
64 bytes from 10.0.0.1: icmp_seq=44788 ttl=64 time=0.062 ms
64 bytes from 10.0.0.1: icmp_seq=44789 ttl=64 time=0.031 ms
64 bytes from 10.0.0.1: icmp_seq=44790 ttl=64 time=0.044 ms
64 bytes from 10.0.0.1: icmp_seq=44791 ttl=64 time=0.031 ms
From 10.0.0.1 icmp_seq=44841 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44842 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44843 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44844 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44845 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44846 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44847 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44848 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44849 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44850 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44851 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44852 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44853 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44854 Destination Host Unreachable
From 10.0.0.1 icmp_seq=44855 Destination Host Unreachable
```

Figure 55 Ping to server

Pings to the server are failing indicating the tcp syn ddos attack was successful. The server's resources have been exhausted to the point of not being able to respond to any connection requests.

4.8.9 Total server failure:

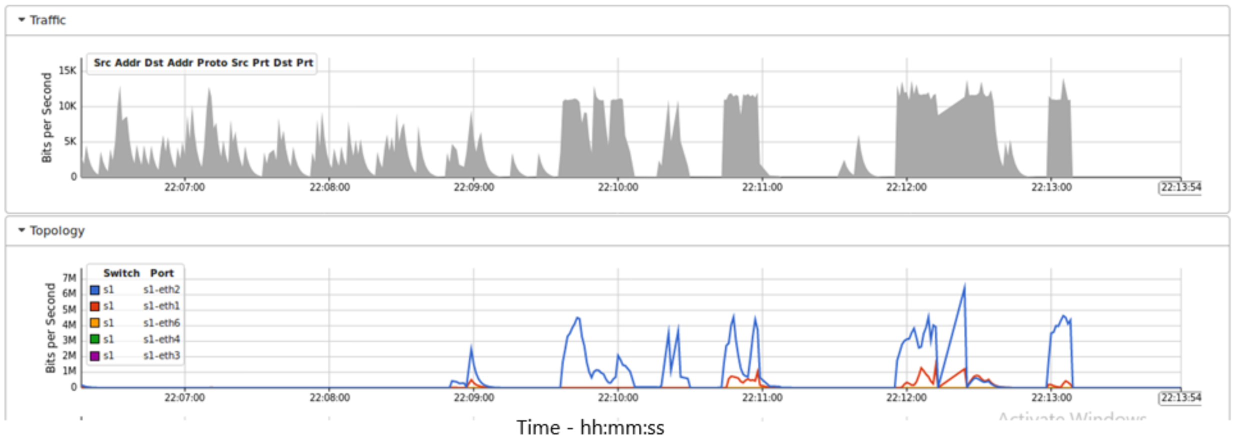


Figure 56 Server failure

Figure 56 shows that there is no traffic coming from the server meaning it has been overwhelmed by the ddos attack.

The SDN was not able to detect the DDoS and left the server exposed leading to its total failure after some time. Services are not available at this point since the web server cannot respond to any connection requests.

5 CHAPTER 5: CONCLUSION

In this research, we sought to find a solution for combatting protocol based DoS and DDoS attacks by taking advantage of the Software Defined Networking architecture. The study has successfully established that SDN can be used to effective detect and stop protocol based DoS/DDoS attacks. For this study, a virtualized environment implemented using Virtualbox, mininet, floodlight controller, sFlow, and OpenFlow switches was used. Nping, hping3 and normal ping were used to generate traffic on the virtual network.

Establish methodologies and technologies that can be used to detect and prevent protocol based DDoS attacks.

This study was set out to identify methodologies and technologies that can be used to detect and stop protocol based DDoS attacks. Taking advantage of the Software Defined Networking architecture where the controller has a view of the whole network and installs flows on the switches in the topology, an application was developed to take advantage of this feature. The following technologies were used to come with a solution to detect and prevent protocol based DDoS attacks:

Open Southbound APIs

OpenFlow and sFlow were used as southbound APIs.

sFlow-RT was used to obtain telemetry streams continuously from the switch and host in the SDN using agents in the switch and hosts. The agents convert these telemetry streams into metrics that were acted upon and accessed via the REST API. This allowed us to define measurements, obtain given metrics, define thresholds, and get notifications. The application built used this information is HTTP/REST calls.

sFlow-RT enabled us to have a view of network performance.

OpenFlow(OF) is a pioneer SDN standard that defines communication protocol enabling the controller to interact with the data plane of devices in the infrastructure layer of the SDN. SDN controllers can send configuration changes down to switches and routers using OpenFlow. Flow tables of the switches are modified by the controller using OpenFlow. For our research, OpenFlow is used to install appropriate flow rules into the switch depending on the state of the network e.g. when an attack is detected, the controller uses OpenFlow to install a flow in the switch instructing it to drop any traffic originating from an IP address identified as a source of malicious traffic.

Controller Operating System

For this research, the Floodlight controller was used since it provides basic connectivity by default. It also allows users to automatically alter the flow rules using the REST API. This capability allows us to create an application that doesn't have to worry about maintaining connectivity thus focusing on optimization of the network.

Open Northbound API

REST API was used as the northbound API. REST (Representational State Transfer) is based on uniform resource identifiers (URIs). URLs are a type of URI and we use them in this research. REST API uses HTTP protocols and JSON thus making it very compatible with browsers. This makes it very simple to deploy and optimal for web applications. REST API was used in this research for client-server communications. REST allowed us to define measurements, obtain given metrics, define thresholds, and get notifications from the underlying layers of the SDN.

Application

The application was built using node.js. Node.js is a free, open source and runs on all platforms i.e. linux, windows, unix, Mac OS etc. It uses javascript on servers. Node.js was chosen because it allows for asynchronous input/output thus supporting higher levels of parallelism thus enabling SDN applications interact with many controllers, monitor systems etc. without blocking hence we get a quick and consistent response time. Node.js has the ability to do the following:

- generation of dynamic page content.
- creation, opening, reading, writing, deletion, and closing files existing on a given server.
- addition, deletion, modification of data stored in a database.

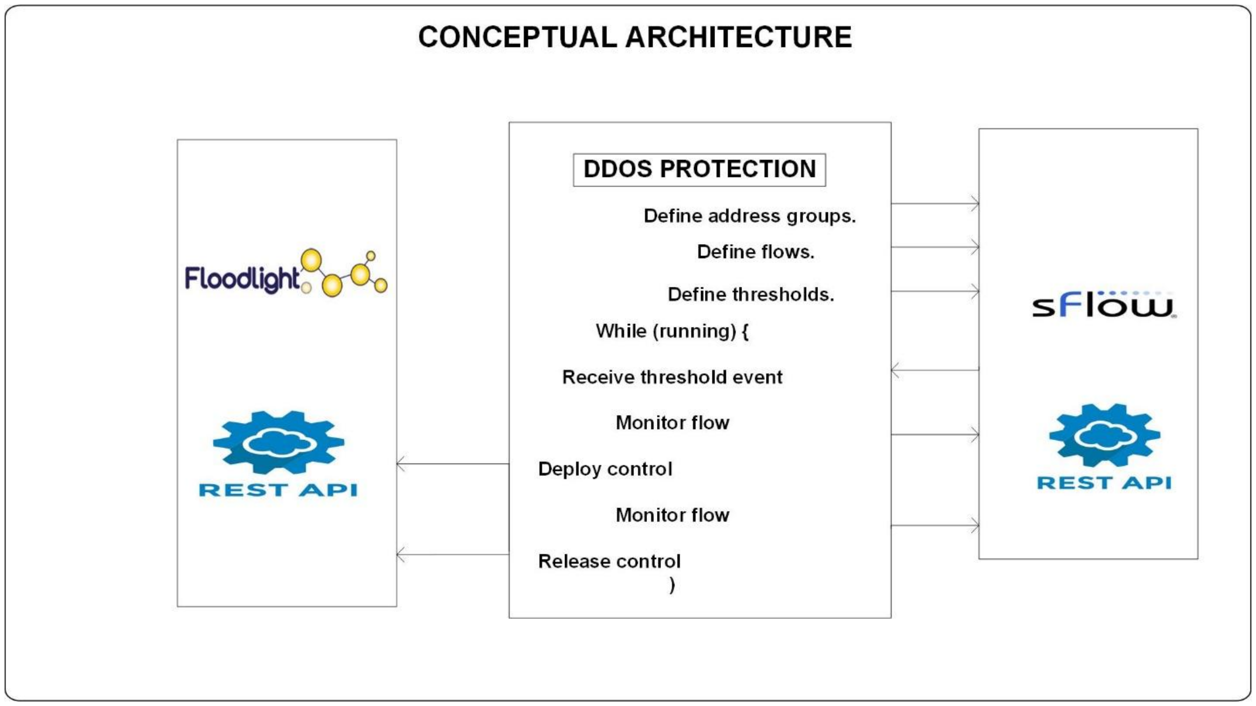
JSON (Javascript Object Notation) was also used in the application. JSON is used to store and transport data because it is lightweight. Most common scenario for this is whenever data is sent to a web page from a server. JSON is constructed using two structures:

- A pair of name/value.
- A listing of values that are ordered.

This is the syntax used to store and exchange data.

JSON is readable by almost all programming languages and can be easily fetched by HTTP.

The methodology used in this research is shown in the diagram below:



9. Definition of address groups – traffic is categorized into internal and external. Internal traffic is represented as an address block and is the address space used on the hosts and switches of the SDN. In our case it’s the address block 10.0.0.0/8. External traffic is any other traffic that might be accessing our network. We define the whole address space as external traffic i.e. 0.0.0.0/0.
10. Flow definition - We define the flows by assigning names to packet characteristics identified and implemented in grouping the same packets into flows or keys, a unique value to associate the flow with and finally a filter for selection of specific traffic.
11. Defining thresholds - We define thresholds of packets that could signal a breach e.g. 10000 packets per second for a given flow.

12. Get threshold events - The application keeps polling for events in order to obtain notifications in case of new events.
13. Monitoring of flows - The flow is monitored and more information regarding the flow is obtained. This information is extracted from the event plus the agent, metrics and data sources.
14. Deploying control – after monitoring and determining that thresholds have been exceeded, the controller installs flows that result in traffic from the identified source being dropped.
15. Monitoring of flows – the flow is continuously monitored to ensure that the control measures took effect on traffic.
16. Releasing control - After a given time, the control action is stopped so that flow table entries engaged in blocking the attacker are released. If the attack happens again, another event will be generated and new control implemented.

Develop an application to be used to detect and prevent protocol based DDoS attacks. The application should detect DDoS in SDN before it overwhelms the controller.

For this research, a node.js application was developed and successfully used to detect and prevent protocol based DDoS attacks. Node.js was chosen because it allows for asynchronous input/output thus supporting higher levels of parallelism thus enabling SDN applications interact with many controllers, monitor systems etc. without blocking hence we get a quick and consistent response time. The application uses the REST API to communicate with the controller which in turn avails all information with regards to the virtual network.

JSON (Javascript Object Notation) was also used in the application. JSON is used to store and transport data because it is lightweight. Most common scenario for this is whenever data is sent to a web page from a server. JSON is constructed using two structures:

- A pair of name/value.
- A listing of values that are ordered.

This is the syntax used to store and exchange data.

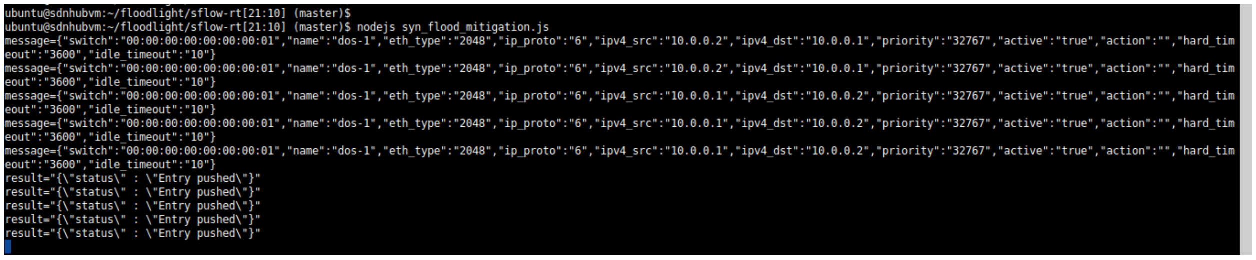
JSON is readable by almost all programming languages and can be easily fetched by HTTP.

Evaluate accuracy of the developed application in terms of detecting and deterring protocol based DDoS attacks.

Through experimentation, the application developed was tested with simulated ping flood and syn flood attacks and on both occasions, the application was able to successfully identify, detect and prevent the attacks.

This was determined by the application output where it was able to stop the actual IP address from which the attack was being simulated.

As shown in the image below, the IP 10.0.0.2 was detected as the source of the TCP SYN attack and consequently stopped from causing further harm.



```
ubuntu@sdnhubvm:~/floodlight/sflow-rt[21:10] (master)$
ubuntu@sdnhubvm:~/floodlight/sflow-rt[21:10] (master)$ nodejs syn_flood_mitigation.js
message={"switch":"00:00:00:00:00:00:00:01","name":"dos-1","eth_type":"2048","ip_proto":"6","ipv4_src":"10.0.0.2","ipv4_dst":"10.0.0.1","priority":"32767","active":"true","action":"","hard_tim
eout":"3600","idle_timeout":"10"}
message={"switch":"00:00:00:00:00:00:00:01","name":"dos-1","eth_type":"2048","ip_proto":"6","ipv4_src":"10.0.0.2","ipv4_dst":"10.0.0.1","priority":"32767","active":"true","action":"","hard_tim
eout":"3600","idle_timeout":"10"}
message={"switch":"00:00:00:00:00:00:00:01","name":"dos-1","eth_type":"2048","ip_proto":"6","ipv4_src":"10.0.0.1","ipv4_dst":"10.0.0.2","priority":"32767","active":"true","action":"","hard_tim
eout":"3600","idle_timeout":"10"}
message={"switch":"00:00:00:00:00:00:00:01","name":"dos-1","eth_type":"2048","ip_proto":"6","ipv4_src":"10.0.0.1","ipv4_dst":"10.0.0.2","priority":"32767","active":"true","action":"","hard_tim
eout":"3600","idle_timeout":"10"}
message={"switch":"00:00:00:00:00:00:00:01","name":"dos-1","eth_type":"2048","ip_proto":"6","ipv4_src":"10.0.0.1","ipv4_dst":"10.0.0.2","priority":"32767","active":"true","action":"","hard_tim
eout":"3600","idle_timeout":"10"}
result="{\"status\" : \"Entry pushed\"}"
result="{\"status\" : \"Entry pushed\"}"
result="{\"status\" : \"Entry pushed\"}"
result="{\"status\" : \"Entry pushed\"}"
result="{\"status\" : \"Entry pushed\"}"
```

Figure 57TCP SYN attack mitigation

Determine whether the application can handle malicious traffic while normal traffic is left unaffected.

The experiments conducted proved that the application can handle malicious traffic while normal traffic is left unaffected.

By singling out specific IP addresses during the attack and coming up with a flow rule that only stops traffic from that particular IP address, other normal traffic is left to continue traversing the network unaffected.

The ddos application singles out only specific hosts and mitigates their destructive nature on the network.

5.1 Future Work

This study was focused on detecting and preventing protocol based DoS/DDoS attacks with ping and syn flood attacks experimented with. A virtualized environment was used.

Further work can be done using real networking equipment. Other protocol attacks can also be studied e.g. DNS attacks, HTTP flooding etc.

Application based DoS/DDoS attacks are also an interesting field that could be explored further.

6 CHAPTER 6: REFERENCES

Cabaj, K., Wytrębowicz, J., Kukliński, S., Radziszewski, P., & Truong , K. D. (2014). SDN Architecture Impact on Network Security. *Federated Conference on Computer Science and Information Systems.* Warsaw.

Carl, G., Kesidis, G., Brooks, R. R., & Rai, S. (2006). Denial-of-service attack-detection techniques. *IEEE Internet Computing*, 82-89.

Dao, N.-N., Park, J., Park, M., & Cho, S. (2015). A Feasible Method to combat against DDoS Attack in SDN Network. *International Conference on Information Networking* (pp. 309-311). Siem Reap, Cambodia: IEEE.

Dittrich, D. (1999, December 31). *University of Washington*. Retrieved from University of Washington: http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt

Floodlight. (2018). Retrieved from Floodlight: http://www.projectfloodlight.org/floodlight/

GitHub. (2017, September 03). Retrieved from GitHub: https://github.com/noxrepo/

GitHub. (2017, September). Retrieved from GitHub: https://github.com/Markus-Go/bonesi

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., Mckeown, N., & Shenker, S. (2008). NOX: Towards an Operating System for Networks. *COMPUTER COMMUNICATION REVIEW, vol 38, no. 3* (pp. 105-110). Association for Computing Machinery (ACM).

Gupta, B. B., Joshi, R. C., & Misra, M. (2009). Defending against Distributed Denial of Service Attacks: Issues and Challenges. *Information Security Journal: A Global Perspective*, 223-248.

Haris, S. H., Ahmad, R. B., & Ghani, M. H. (2010). Detecting TCP SYN Flood Attack based on Anomaly Detection. *2010 Second International Conference on Network Applications, Protocols and Services* (pp. 240-244). Washington, DC, USA: IEEE Computer Society.

Hussain, A., Heidemann, J., & Papadopoulos, C. (2003). A Framework for Classifying Denial of Service Attacks. *Applications, technologies, architectures, and protocols for computer communications* (pp. 100-110). Karlsruhe, Germany: ACM.

INCAPSULA. (2017, September). Retrieved from INCAPSULA: https://www.incapsula.com/ddos/ddos-attacks/

inMon. (2018, 11 11). Retrieved from inMon: https://inmon.com/products/sFlow-RT.php

J. , S.-T., & Cristianini, N. (2000). *Support Vector Machines and other kernel-based learning methods.* Cambridge University Press.

Kesavan, A. (2016, November 15). *ThousandEyes*. Retrieved from ThousandEyes: https://blog.thousandeyes.com/three-types-ddos-attacks/

Kohonen, T. (1997). *Self-organizing Maps.* Secaucus, NJ, USA: Springer-Verlag.

Linux Foundation. (2017). Retrieved from OPENDAYLIGHT: http://www.opendaylight.org

Luo, S., Wu, J., Li, J., & Pei, B. (2015). A Defense Mechanism for Distributed Denial of Service Attack in Software-Defined Networks. *2015 Ninth International Conference on Frontier of Computer Science and Technology.* Dalian, China: IEEE.

Mininet. (2017). Retrieved from Mininet: http://mininet.org

Mirkovic, J., Martin, J., & Reiher, P. (2010). A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms. *Data Privacy Management and Autonomous Spontaneous Security* (pp. 220-236). Springer.

Mousavi, S. M. (2014). *Early Detection of DDoS Attacks in Software Defined Networks Controller.*

netem. (2018). Retrieved from netem: https://wiki.linuxfoundation.org/networking/netem

NOX. (2018). Retrieved from NOX: https://github.com/noxrepo/nox

ns-3. (2018). Retrieved from ns-3: https://www.nsnam.org/

ONF. (2017). *Open Networking Foundation*. Retrieved November 10, 2017, from https://www. opennetworking.org

ONOS. (2018). Retrieved from ONOS: https://onosproject.org/

Open Networking Foundation (ONF). (2017, November 10). Retrieved from Open Networking Foundation (ONF): https://www. opennetworking.org

OpenDaylight. (2018). Retrieved from OpenDaylight: https://www.opendaylight.org/

Pescatore, J. (2014, February). *SANS.* Retrieved from SANS: https://www.sans.org/reading-room/whitepapers/analyst/ddos-attacks-advancing-enduring-survey-34700

Phan, T. V., Bao, K. N., & Park, M. (2016). A Novel Hybrid Flow-based Handler with DDoS Attacks in Software-Defined Networking. *Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress.* Seoul: IEEE.

Python Standard Library. (2017, September). Retrieved from Python Standard Library: https://docs.python.org/2/library/random.html

Rana, D. S., Garg, N., & Chamoli, S. K. (2012). A Study and Detection of TCP SYN Flood Attacks with IP spoofing and its Mitigations. *International Journal of Computer Technology and Applications*.

Rodrigo, B., Edjard, M., & Alexandre, P. (2010). Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow. *35th Annual IEEE Conference on Local Computer Networks*, (pp. 405-416). Denver, Colorado.

Rodrigo, B., Edjard, M., & Alexandre, P. (2010). Light-weight Distributed DoS Flood Attack Detection Using NOX & OpenFlow. Denver, Colorado: 35th Annual IEEE Conference on Local Computer Networks.

Scapy. (2017, September). Retrieved from Scapy: http://www.secdev.org/projects/scapy *sflow*. (2017, September). Retrieved from sflow: http://www.sflow.org

sflow. (2017, September). Retrieved from sflow: http://blog.sflow.com/2011/01/presentation.html

Srivastava, A., Gupta, B. B., Tyagi, A., Sharma, A., & Mishra, A. (2011). A Recent Survey on DDoS Attacks and Defense Mechanisms. *Advances in Parallel Distributed Computing* (pp. 570-580). Springer.

Sutter, J. (2009, August 6). *CNN.* Retrieved from CNN: http://www.cnn.com/2009/TECH/08/06/twitter.attack/index.html

Virtualbox. (2018). Retrieved from Virtualbox: https://www.virtualbox.org/

Wesley M. Eddy, V. F. (2006, December). *Cisco.* Retrieved from Cisco: https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-34/syn-flooding-attacks.html