“DDoS Attack Detection and Mitigation in SDN”

A

Project Report

submitted in partial fulfillment of the

requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

SCHOOL OF COMPUTER SCIENCE ENGINEERING

by

Name	**Roll No.**
Apurv Ranjan	**R111214008**
Mohit Kumar Jaiswal	**R111214073**
Abhinav Yadav	**R111214068**
Deepak Bhist	**R111214017**

under the guidance of

Dr. Inder Singh (S.G.)

Assistant Professor

SoCSE, UPES

Department of Computer Science & Engineering

School of Computer Science Engineering

University of Petroleum & Energy Studies

Bidholi, Via Prem Nagar, Dehradun, UK

May – 2017

CANDIDATE's DECLARATION

We hereby certify that the project work entitled **"DDoS Attack Detection and Mitigation in SDN"** in partial fulfillment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in IT INFRASTRUCTURE MANAGEMENT and submitted to the Department of Computer Science & Engineering at School Of Computer Science Engineering, University of Petroleum & Energy Studies, Dehradun, is an authentic record of our work carried out during a period from **January, 2017** to **May, 2017** under the supervision of **Dr. Inder Singh(S.G.), Assistant Professor, SoCSE, UPES.**

The matter presented in this project has not been submitted by us for the award of any other degree of this or any other University.

Name	**Deepak Bisht**	**Apurv Ranjan**	**Mohit Kumar Jaiswal**	**Abhinav Yadav**
Roll No.	**R111214017**	**R111214008**	**R111214073**	**R111214068**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: 18th May 2017

Dr. G. Hanumat Sastry
Head-Department of Systemics
Program Head of B. Tech CSE (IT)
School of Computer Science Engineering
University of Petroleum & Energy Studies
Dehradun – 248 001 (Uttarakhand)

Dr. Inder Singh (S.G.)
Assistant Professor
SoCSE, UPES
Project Guide

ACKNOWLEDGEMENT

We wish to express our deep gratitude to our guide **Dr. Inder Singh (S.G)** for all advice, encouragement and constant support he has given us throughout our project work. This work would not have been possible without his support and valuable suggestions.

We sincerely thank to our respected Program Head of the Department, **Dr. G. Hanumat Sastry** for his great support in doing our project in **Area** at **SoCSE**.

We are also grateful to **Dr. Manish Prateek,** Director and **Dr. Kamal Bansal** Dean CoES, UPES for giving us the necessary facilities to carry out our project work successfully.

We would like to thank all our **friends** for their help and constructive criticism during our project work. Finally, we have no words to express our sincere gratitude to our **parents** who have shown us this world and for every support they have given us.

Name	**Deepak Bisht**	**Apurv Ranjan**	**Mohit Kumar Jaiswal**	**Abhinav Yadav**
Roll No.	**R111214017**	**R111214008**	**R111214073**	**R111214068**

ABSTRACT

Software-defined networking (SDN) is an approach to computer networking that allows network administrators to manage network services and provides central control over the network. This control works as an operating system that can send instructions and apply changes through its interface. This operating system is called controller.
The SDN architecture decouples the network architecture into two planes, which are known as Control plane and Data plane, this decoupling of architecture enables control plane to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. Although central control is the major advantage of SDN, it is also a single point of failure if it is made unreachable by a Distributed Denial of Service Attack (DDoS).

The main objective of this research are utilizing the central control of SDN for attack detection and proposing a solution that is effective and light weighted in terms of the resources that it uses. This paper shows how DDoS attacks can exhaust controller resources and provides a solution to detect such attacks by calculating entropy variation of source the IP address.

Keywords: SDN, DDoS, IP, OpenFlow, Floodlight Controller, Entropy, etc.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
1. INTRODUCTION

1.1 HISTORY

Software-defined networking (SDN) is an approach to computer networking that allows network administrators to manage network services through abstraction of lower-level functionality. This is done by decoupling the system that makes decisions about where traffic is sent (the control plane) from the underlying systems that forward traffic to the selected destination (the data plane). The inventors and vendors of these systems claim that this simplifies networking.

Software Defined Networks (SDN) provide a new and novel way to manage networks. In SDN, switches do not process incoming packets - they simply look for a match in their forwarding tables. If there is no match, packets are sent to the controller for processing. The controller is the operating system of SDN. It processes the packets and decides whether the packets will be forwarded in the switch or dropped. By applying this procedure, SDN separates the forwarding and the processing planes. If the connection between the switches and the controller is lost, the network will lose its processing plane. This means that packet processing is no longer done in the controller and by losing the controller, the SDN architecture is lost.

One of the possibilities that can cause the controller to be unreachable is a Distributed Denial of Service (DDoS) attack. In DDoS attacks, a large number of packets are sent to a host or a group of hosts in a network. If the source addresses of the incoming packets are spoofed, which is usually the case, the switch will not find a match and has to forward the packet to the controller. The collection of legitimate and DDoS spoofed packets can bind the resources of the controller into continuous processing up to the point where they are completely exhausted. This will make the controller unreachable for the newly arrived legitimate packets and may bring the controller down causing the loss of the SDN architecture. Even if there is a backup controller, it has to face the same challenge.

There are many types of attacks crafted specially for:

- Congesting network resources,
- Draining CPU memory,
- Reducing computing power,
- Exploiting timers,
- Poisoning domain name translations etc.

There are also attacks that could be carried out at application level, hindering the normal functioning of a service. There are attacks that are designed to crash a web browser, email application or even a media player. When a specific application is disrupted and when normal functioning is hindered, it is called the Application Level Denial of Service.

As a worst case scenario, there are attacks that can cause permanent damage to a system. These kinds of attacks are called the Permanent Denial of service or Plashing. Permanent Denial of Service attacks are mostly firmware based that aims at completely destroying the hardware. Firmware’s are the inbuilt code or program that is embedded on every electronic system for its proper functioning. When an attacker is able to change the firmware and replace it with a defective or corrupt one, the hardware could no longer be used. These attacks could be directed towards networking components like routers, switches or bridges and thus bringing an entire routing table to collapse. A fault in a single router might lead to a huge outage if it does not have enough backups and rerouting. Often devices, who try to upgrade their firmware online without checking for the signature of a trusted source, fall prey for this attack.

1.2. DENIAL OF SERVICE ATTACK MECHANISMS:

Denial of service attacks are further classified into many categories according to the style with which it is implemented. The following paragraphs discuss few of the most wellknown categories.

1.2.1. Distributed Denial of service:

Distributed Denial of service has the cohesive strength of many compromised systems working towards a single cause. The first stage of this attack is to build its platform with many host systems that can work under remote commands. The attacker group would first scan networks to hunt for vulnerable systems that are weak in security features. According to researchers there are millions of host machines that are vulnerable without secure patches and proper updates that often fall victims to these attackers. Once the scanning procedure is completed, attackers would bring these hosts into control using software exploitations like buffer over flow, dangling pointers, code injection etc. Special root kits are also used in many cases that are installed in a host system to incur these software exploitations. After having sufficient hosts under control, attackers also create backdoors that allows special access that is used for future entry. The attackers also update the hosts and tighten its security so that another attacker does not use the same host. Any future entry would be done using the back entry that has been specially crafted.

1.2.2. Low-rate TCP targeted Denial of Service:

Unlike the Distributed Denial of Service, low-rate TCP targeted attacks does not employ numerous packets to flood the network. Instead, it exploits the working mechanism of TCP timers thus bringing the throughput of a system to almost zero. These low-rate attacks are crafted to generate packets only periodically in very minimal quantity. Thus the attacking packets can easily disguise with the legitimate packets and escape from the Anti-Dos traffic monitoring systems. The attacks carried out this way exploiting the TCP timers are coined with the term "shrew attacks". It is also indispensable to understand the TCP working procedure before discussing this attack during congestion in TCP, the congestion window is gradually reduced until the network is clear.

Thus during congestion, the sender's rate is reduced which apparently reduces the potential throughput. The TCP waits for the Retransmission Time Out (RTO) to expire after which the data is sent again. When the congestion is more, the RTO timer is doubled after which the packets are retransmitted. Thus during a low rate attack, when packets are lost, TCP enters RTO. When an attacker is able to calculate this RTO time and sends attacking packets to create packet collision and loss, the attacker can push the TCP into waiting state. Hence, there is no need for flooding the network with packets, but only send packets when the timer is about to expire and push it again into the RTO waiting time. This type of attack can effortlessly escape the traffic monitors due to its low traffic rate and is a serious challenge for the security experts.

1.3. WHAT DOES ATTACKER WANT?

There are several reasons why an attacker would like to cause DDoS. It could be a group of people who would like to bring down a specific webpage or website in order to keep it isolated from the business. Thus the company might lose all its online transactions and thus end up failing. Rivalry in business could be one main factor for these attacks. There have also been incidents where protestors show their dislike with an attack. This is often done when attackers target a government website and bring it down. Examples would definitely include the attacks carried out against the Georgian national, finance and president's websites. There have also been similar attacks on Iranian websites as a part of election protest. Rioters chose these attacks against government and public websites as there is minimal chances of being caught. Another reason which aids to these attacks is the simplicity involved. Any beginner could perform this attack effortlessly without having much technical expertise. Attackers often post their attacking tools and scripts online to aid others who like to carry out similar operation. There are websites and forums that give out tools along with instruction manual that makes easier for anyone to carry out such attacks. People who carry out attacks without having actual knowledge about it are called the 'Script Kiddies'.

1.3.1. A Study of past DoS occurrences:

There are many occurrences that have been happening since the last ten years and there is still no effective control for this attack. One of the most talked about attack happened in the year 2000, February 7 when yahoo servers were crashed. The famous internet site was unavailable for several hours which affected the business of yahoo considerably. Buy.com, ebay and CNN were the other giant companies that were attacked, the very next day after yahoo. E-bay is an online bank that undertakes millions of transactions online. The site was completely inaccessible which incurred huge loss to the company. The downtime was calculated as three hours for yahoo and the other websites were down for several more hours. There were also other companies like ZDnet, Etrade and Excite that were bombarded with more than 1 gigabit per second of data which made the server isolated from legitimate requests.

Apart from using DoS attacks as a single weapon against the companies, attackers use it along with other destructive security vulnerabilities. One of the most interesting incidents took place in the year 2008, July 2 when Revolutionary Armed Forces of Colombia was attacked using Denial of

Service and Man in the Middle attacks (MITM). The attackers were later found to be Colombian National Forces (CNF) who carried out series of DoS and MITM attacks in order to bring out their hostages. Revolutionary Armed Forces of Colombia also called FARC had to release fifteen most crucial hostages including Betancourt Ingrid (a famous social leader), 3 American citizens and 11 other members. This is one of the most talked about incident in the history of security industry that freed several hostages without a single arm or ammunition being involved.

1.4. DENIAL OF SERVICE IN DETAIL

In computing, a denial-of-service (DoS) attack is an attempt to make a machine or network resource unavailable to its intended users, such as to temporarily or indefinitely interrupt or suspend services of a host connected to the Internet. DDoS is defined as distributed denial of service. A malicious hacker uses a DDoS attack to make a computer resource (i.e. – website, application, e-mail, voicemail, network) stop responding to the registered users. The malicious hacker does this by commanding a more than one remotely-controlled computers to send a flood of network traffic to the target. The target becomes so busy dealing with the attacker's requests that it doesn't have time to respond to registered users' requests. How DDoS works is by causing the target system to stop responding, resulting in long delays and outages.

DDoS attacks can be divided in the two groups: semantic attacks and brute-force (flood) attacks. A semantic attack exploits a specific feature or implementation bug of some protocol or application installed at the victim in order to consume excess amounts of its resources. For example, an attacker can send a specific sequence of packets initiating CPU time consuming procedures on the server. In case of a large number of such requests, the victim is unable to handle requests from legitimate clients. Undesirable impact from such attack can be minimized by protocol or software modifying and by applying of special filter mechanisms.

1.4.1. ATTACK TECHNIQUES:

1.4.1.1. Network Based Attacks

TCP SYN Flooding:

DoS attacks often exploit stateful network protocols (Jian 2000, Shannon et al. 2002), because these protocols consume resources to maintain states. TCP SYN flooding is one of such attacks and had a wide impact on many systems. When a client attempts to establish a TCP connection to a server, the client first sends a SYN message to the server. The server then acknowledges by sending a SYN-ACK message to the client. The client completes the establishment by responding with an ACK message. The connection between the client and the server is then opened, and the service-specific data can be exchanged between them. The abuse arises at the half-open state when the server is waiting for the client's ACK message after sending the SYN-ACK message to the client. The server needs to allocate memory for storing the information of the half-open connection. The memory will not be released until either the server receives the final ACK message or the half-open connection expires.

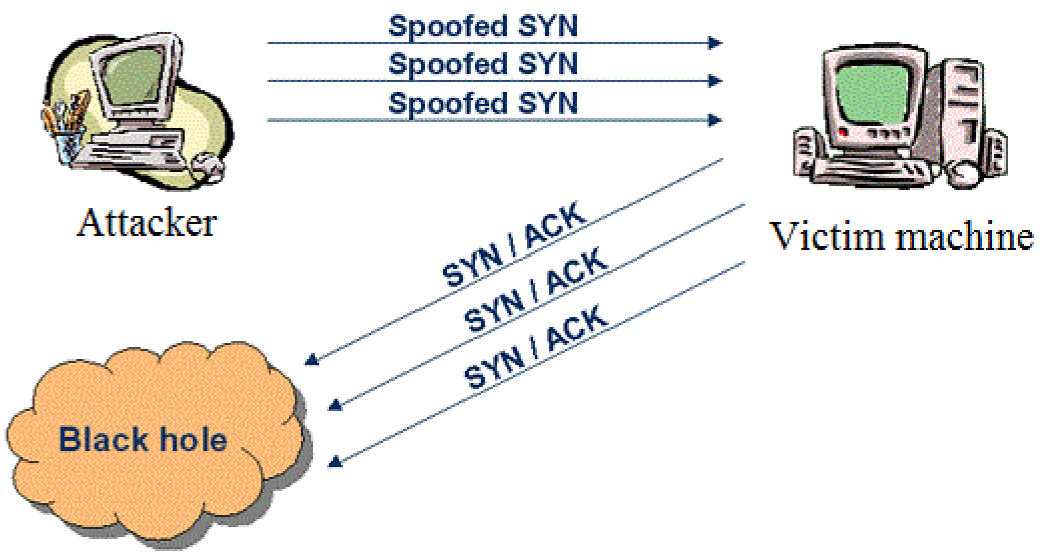
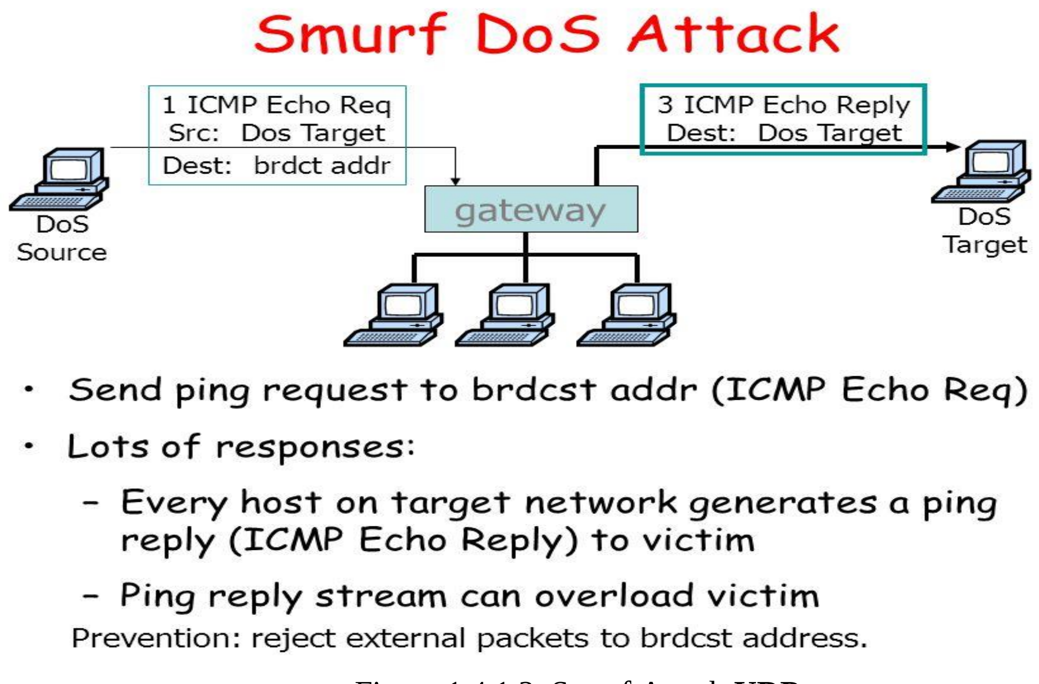


Figure 1.4.1.1. DDoS Syn Flood

Attacking hosts can easily create half-open connections via spoofing source IPs in SYN messages or ignoring SYN-ACKs. The consequence is that the final ACK message will never be sent to the victim. Because the victim normally only allocates a limited size of space in its process table, too many half-open connections will soon fill the space. Even though the halfopen connections will eventually expire due to the timeout, zombies can aggressively send spoofed TCP SYN packets requesting connections at a much higher rate than the expiration rate. Finally, the victim will be unable to accept any new incoming connection and thus cannot provide services.

ICMP Smurf Flooding:

ICMP is often used to determine if a computer in the Internet is responding. To achieve this task, an ICMP echo request packet is sent to a computer. If the computer receives the request packet, it will return an ICMP echo reply packet. In a smurf attack, attacking hosts forge ICMP echo requests having the victim's address as the source address and the broadcast address of these remote networks as the destination address (CERT 1998). If the firewall or router of the remote network does not filter the special crafted packets, they will be delivered (broadcast) to all computers on that network. These computers will then send ICMP echo reply packets back to the source (i.e., the victim) carried in the request packets. The victim’s network is thus congested.



- Send ping request to brdcst addr (ICMP Echo Req)
- Lots of responses:
  - Every host on target network generates a ping reply (ICMP Echo Reply) to victim
  - Ping reply stream can overload victim

Prevention: reject external packets to brdcst address.

Figure 1.4.1.2. Smurf Attack **UDP**

Flooding:

By patching or redesigning the implementation of TCP and ICMP protocols, current networks and systems have incorporated new security features to prevent TCP and ICMP attacks. Nevertheless, attackers may simply send a large amount of UDP packets towards a victim. Since an intermediate network can deliver higher traffic volume than the victim network can handle, the flooding traffic can exhaust the victim's connection resources. Pure flooding can be done with any type of packets. Attackers can also choose to flood service requests so that the victim cannot handle all requests with its constrained resources (i.e., service memory or CPU cycles). Note that UDP flooding is similar to flash crowds that occur when a large number of users try to access the same server simultaneously. However, the intent and the triggering mechanisms for DDoS attacks and flash crowds are different.

Intermittent Flooding:

Attackers can further tune their flooding actions to reduce the average flooding rate to a very low level while achieving equivalent attack impacts on legitimate TCP connections. In shrew attacks, attacking hosts can flood packets in a burst to congest and disrupt existing TCP connections. Since all disrupted TCP connections will wait a specific period (called retransmission-time-out (RTO)) to retransmit lost packets, attacking hosts can flood packets at the next RTO to disrupt retransmission. Thereby, attacking hosts can synchronize their flooding at the following RTOs and disable legitimate TCP connections. Such collaboration among attacking hosts not only reduces overall flooding traffic, but also helps avoid detection. Similar attack techniques targeting services with congestion control mechanisms for Quality of Service (QoS) have been discovered by Guirguis et al. (2005). When a QoS enabled server receives a burst of service requests, it will temporarily throttle incoming requests for a period until previous requests have been processed. Thus, attackers can flood requests at a pace to keep the server throttling the incoming requests and achieve the DoS effect. Guirguis's study showed that a burst of 800 requests can bring down a web server for 200 seconds, and thereby the average flooding rate could be as low as 4 requests per second.

1.4.1.2. Host Based Attacks

Besides misusing network protocols, attackers can also launch DoS attacks via exploiting vulnerabilities in target's applications and systems. Different from network based attacks, this type of attacks is application specific, i.e., exploiting particular algorithms memory structure, authentication protocols, implementation etc. Attacks can be launched either from a single host as a conventional intrusion or from a number of hosts as a network based DDoS attack. The traffic of host based attacks may not be as high as network based attacks, because application flaws and deficiencies can easily crash applications or consume a tremendous amount of computer resources.

Several example attacks are described as follows:

Dean et al. (2001) identified that attackers could easily arrange an attack such that E-commerce web sites remain available, but clients are unable to complete any purchase. Such an attack is based on going after the secure server that processes credit card payments. In such E-commerce applications, the SSL/TLS protocol is used to make secure connections between clients and servers. The protocol allows a client to request the server to perform an RSA decryption. RSA decryption is an expensive operation. For instance, a large secure web site can process a few thousand RSA decryptions per second. If an SSL handshake request takes 200 bytes and a server can process 5000 decryptions per second, 1MB/s of requests is sufficient to paralyze an E-commerce site, which is a hard-to-notice small amount of traffic. Attackers can also send large modulo values via client certificates to increase the RSA computation per authentication. Consequently, mutual authentication cannot be done quickly and service performance is downgraded.

1.4.2. Attack Network

Many recent DoS attacks (also called DDoS attacks) were launched from distributed attacking hosts. A DDoS attack is launched in two phases. First, an attacker builds an attack network which is distributed and consists of thousands of compromised computers (called zombies, bots, or attacking hosts). Then, the attacking hosts flood a tremendous volume of traffic towards victims either under the command of the attacker or automatically. To build an attack network, the attacker looks for computers that are poorly secured, such as those not having been properly patched. In general, a vulnerable host can be compromised via two types of approaches. One is to entice users to run malicious programs, such as a virus, a spyware, or a Trojan horse carried in malicious emails, files, or web pages. The other approach is via automated malicious programs; such as worms that can automatically scan vulnerable remote computers. The vulnerability in these computers is then exploited to allow the attacker to break into and install DoS attacking programs that further scan other hosts, install backdoors and flood packets. The attacker is thus called the master of these compromised computers (zombies).

Some DoS programs have the ability to register the compromised computer as a member in the attack network controlled by the attacker. In addition, the newly compromised computers will automatically repeat the scanning and exploiting process to look for other vulnerable computers. Because of the self-propagation, a large attack network can quickly be built to include hundreds or thousands of computers.

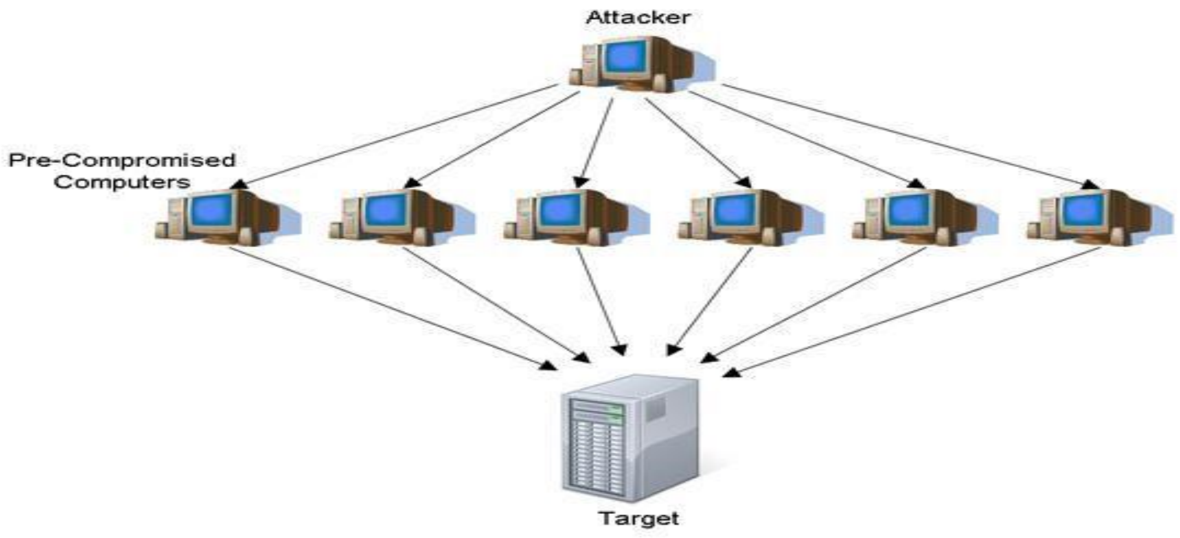


Figure 1.4.2.1. DDoS Attacked Network

1.4.3. Why a DoS/DDoS Attack May Succeed

The design of the Internet is one of the fundamental reasons for successful DoS attacks. The Internet is designed to run end-to-end applications. Routers are expected to provide the best-effort packet forwarding, while the sender and the receiver are responsible for achieving desired service guarantees such as quality of service and security. Accordingly, different amounts of resources are allocated to different roles. Routers are designed to handle large throughput that leads to the design of high bandwidth pathways in the intermediate network.

On the contrary, end hosts may be only assigned as much bandwidth as they need for their own applications.

Consequently, each end host has less bandwidth than routers. Attackers can misuse the abundant resources in routers for delivery of numerous packets to a target. The control and management of the Internet is distributed. Each component network is run according to local policies designed by its owners. No deployment of security mechanisms or security policy can be globally enforced. Because DoS attacks are commonly launched from systems that are subverted through security-related compromises, the susceptibility of the victim to DoS attacks depends on the state of security in the rest of the global Internet, regardless of how well the victim may be secured. Furthermore, it is often impossible to investigate cross-network traffic behaviours in such a distributed management. If one party in a two-way communication (sender or receiver) misbehaves, it can cause arbitrary damage to its peer. No third party will step in to stop it.

The design of the Internet also hinders the advancement of DDoS defences. Many solutions have been proposed to require routers and source networks, together with victims, to participate in constructing a distributed and coordinated defence system. However, the Internet is administered in a distributed manner. Besides the difference of individual security policies, many entities (source networks and routers) in the Internet may not directly suffer from DDoS attacks. Hence, they may not have enough incentive to provide their own resources in the cooperative defence system.

Finally, the scale of the Internet requires more effort to get a defence system benchmark and test bed to evaluate and compare proposed defence technologies. Defence systems should be evaluated with large-scale experiments, data should be collected from extensive simulations, and cooperation is indispensable among different platforms and networks. Recently, researchers, industries and the US government are trying to develop a large-scale cyber security test bed and design benchmarking suites and measurement methodology for security systems evaluation.

1.5. REQUIREMENT ANALYSIS

1.5.1. Hardware Requirements

- 8 GB RAM
- 100 GB Hard Disk

1.5.2. Software Requirements

- Ubuntu 14.04(LTS)
- Mininet
- Virtual Machine
- Floodlight Controller
- JDK
- Eclipse
- Scapy

1.6. OBJECTIVE

1.6.1. MAIN OBJECTIVE

The objectives of this project are broken down into smaller categories to make it effective and achievable.

- The first goal would be to understand the subject in detail by taking into consideration the previous incidents and attacks that happened in the past. IEEE papers and related white papers were carefully chosen to understand this subject matter in depth.
- The second objective is DDoS attack is defined as distributed denial of service which can directly affect the performance of SDN network. So, our main objective in our project is to detect the DDoS attacks and apply a DDoS defence mechanism that aims to filter all TCP traffic issued by unauthorized clients on network level. Therefore, unauthorized malefactor is unable to perform semantic attacks based on TCP protocol on the victim destination.
- The third objective would be set up a test bed environment and choose tools that are required to carry out live implementation. Since carrying out an attack in a real time environment needs extra care when compared to simulation, a dedicated mininet virtual environment was chosen. There are various tools and codes that are capable of causing Denial of Service. Such an attacking tool is selected here for this project work. This project also demands proper ways to measure the attack when it is actually occurring. Apart from these, this project also aims at suggesting possible solutions in mitigating this attack. These attacks should be able to save the victim and also provide access to legitimate clients who would require service during an attack. Summing up all, this project objective would be to investigate and learn subject in depth, implementing the attacks, identifying and measuring the attack and finally adopting counter measures to defend Denial of Service attack.

1.6.2. SUB OBJECTIVES

- Mininet installed and running
- Learn and implement various network topologies using SDN
- Install and configure Floodlight controller
- Start with a Floodlight module on eclipse Generate traffic using scapy
- Apply techniques to detect DDoS attack

CHAPTER 2

2. SYSTEM ANALYSIS

2.1. EXISTING SYSTEM

Operating and maintaining a computer network is an arduous task. To express the required high-level network policies, network operators need to configure each individual network device separately - from a heterogeneous collection of switches, routers, middle boxes, etc. - using vendor specific and low-level commands. In addition to configuration complexity, networks are dynamic, and operators have little or no mechanisms to automatically respond to network events. It is therefore difficult to enforce the required policies in such a continually changing environment. Traditional networks are managed through low-level and vendorspecific configurations of individual network components, which is a very complicated and error-prone process. And nowadays computer networks are becoming increasingly complex and difficult to manage. This increases the need for a general management paradigm that provides common management abstractions, hides the details of the physical infrastructure, and enables flexible network management. Making the network programmable leads to such a general paradigm, as programmability simplifies network management and enables network innovations.

Software-defined networking (SDN) is an approach to computer networking that allows network administrators to manage network services through abstraction of lower-level functionality. This is done by decoupling the system that makes decisions about where traffic is sent (the control plane) from the underlying systems that forward traffic to the selected destination (the data plane). The inventors and vendors of these systems claim that this simplifies networking.

Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications.

Software Defined Networking (SDN) has been proposed to enable programmable networks. In SDN, the network is considered to have two components:

 Control plane which determines how to handle and forward data traffic, and
Data plane which handles and forwards data traffic toward its destination.

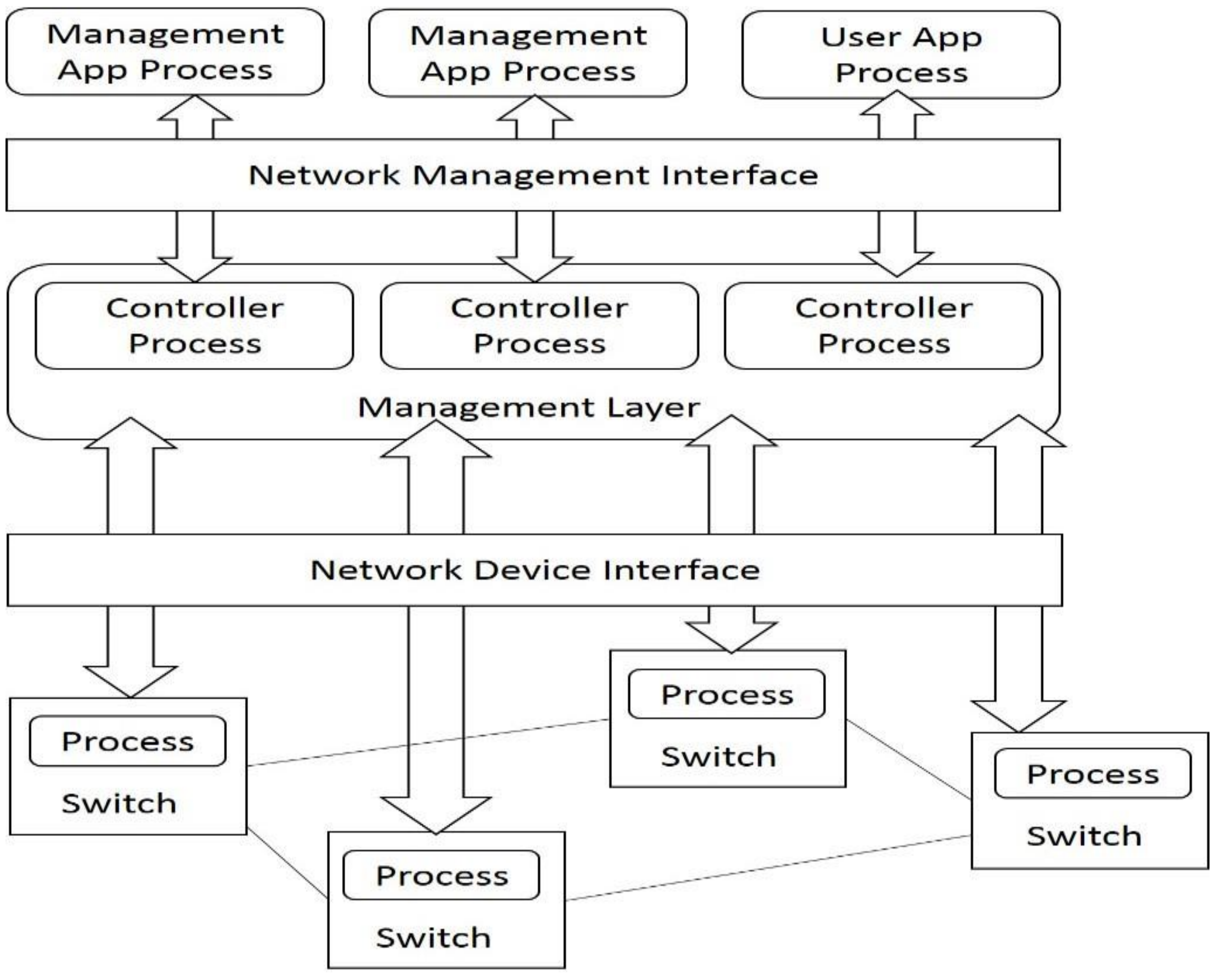


Figure 2.1.1 SDN Functionality of each layer

SDN separates the control plane and data plane, and focuses on programming the control plane through a network management layer. Through a high-level interface provided by the network management layer, network managers can easily manage the network without dealing with the complexity of low-level network details. In an SDN network, the Network Device Interface can be supported by any mechanism that provides communication between the control plane (management layer) and data plane (switch processes). Open Flow is such a protocol that gives the management layer access to switches and routers. With the separation of the control plane from the data plane that lays the ground to the Software Defined Networking paradigm, network switches become simple forwarding devices and the control logic is implemented in a logically centralized controller though in principle physically distributed. In SDN, the controller is the entity that dictates the network behaviour. The logical centralization of the control logic in a software module that runs in a standard server, the network operating system offers several benefits. First, it is simpler and less error-prone to modify network policies through software, than via low-level device configurations. Second, a control program can automatically react to spurious changes of the network state and thus maintain the high-level policies in place. Third, the centralization of the control logic in a controller with global knowledge of the network state simplifies the development of more sophisticated network functions.

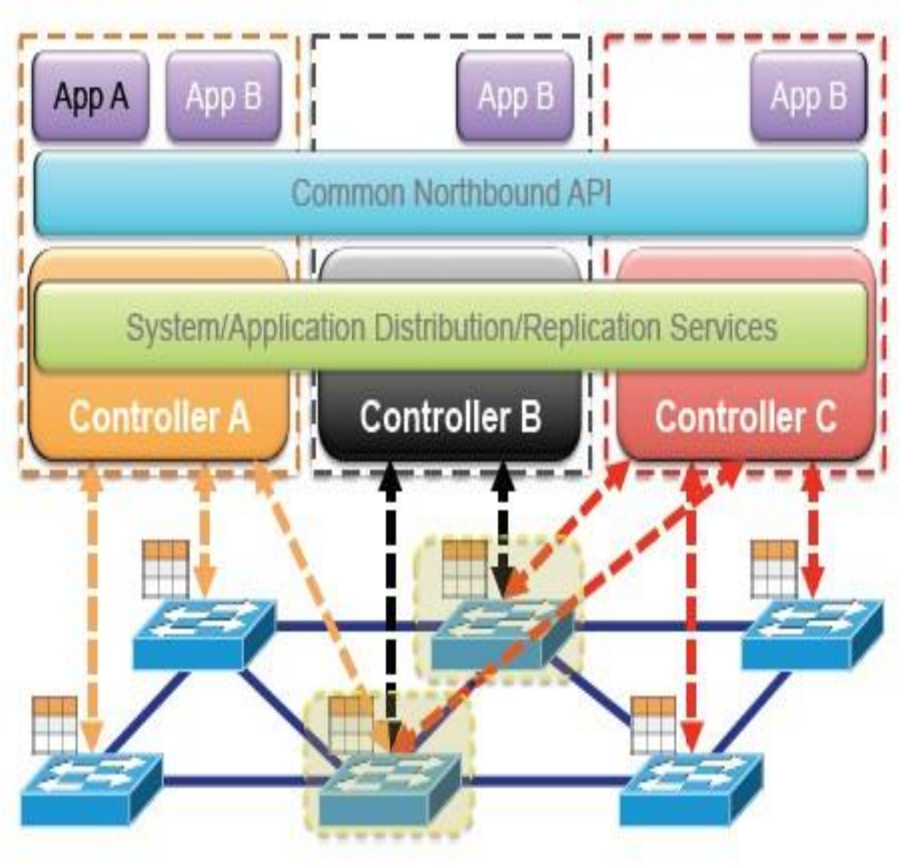


Figure 2.1.2 SDN Architecture

2.1.1. SDN OPERATIONS:

As shown in Figure 2.1.2, the SDN devices contain forwarding functionality for deciding what to do with each incoming packet. The devices also contain the data that drives those forwarding decisions. The data itself is actually represented by the flows defined by the controller, as depicted in the upper-left portion of each device. A flow describes a set of packets transferred from one network endpoint (or set of endpoints) to another endpoint (or set of endpoints). The endpoints may be defined as IP address- TCP/UDP port pairs, VLAN endpoints, layer three tunnel endpoints, and input ports, among other things. One set of rules describes the forwarding actions that the device should take for all packets belonging to that flow.

A flow is unidirectional in that packets flowing between the same two endpoints in the opposite direction could each constitute a separate flow. Flows are represented on a device as a flow entry.

A flow table resides on the network device and consists of a series of flow entries and the actions to perform when a packet matching that flow arrives at the device. When the SDN device receives a packet, it consults its flow tables in search of a match. These flow tables had been constructed previously when the controller downloaded appropriate flow rules to the device. If the SDN device finds a match, it takes the appropriate configured action, which usually entails forwarding the packet. If it does not find a match, the switch can either drop the packet or pass it to the controller, depending on the version of OpenFlow and the configuration of the switch.

- **Southbound API**

The interaction between the controller and the deployed device is commonly referred to as southbound application programing interface (API). A common, open standard SDN protocol, and one of the most popular options for southbound APIs is OpenFlow.

- **Northbound API**

The opposite of the southbound API is the northbound API, which is labelled in Figure 1 as just API; it involves the communications from a business application to the controller. This is the point where the real meat and potatoes exists for SDN. Anybody with basic knowledge of programming can create an SDN application. The SDN application itself can be written to perform a very large set of potential activities; the most basic of which could be to tell a deployed device to forward a packet with a matching set of contents out a specific interface. However, the northbound API can be very complex, like providing a Quality of Service provisioned path from one end of a network to another dynamically, then tearing it back down when the application was completed with its network interaction.

The SDN controller is responsible for abstracting the network of SDN devices it controls and presenting an abstraction of these network resources to the SDN applications running above. The controller allows the SDN application to define flows on devices and to help the application respond to packets that are forwarded to the controller by the SDN devices. Since one controller can control a large number of network devices, these calculations are normally performed on a high-performance machine with an order-of-magnitude performance advantage over the CPU and memory capacity than is typically afforded to the network devices themselves. For example, a controller might be implemented on an eight-core, 2-GHz CPU versus the single-core, 1-GHz CPU that is more typical on a switch.

SDN applications are built on top of the controller. These applications should not be confused with the application layer defined in the seven-layer OSI model of computer networking. Since SDN applications are really part of network layers two and three, this concept is orthogonal to that of applications in the tight hierarchy of OSI protocol layers. The SDN application interfaces with the controller, using it to set proactive flows on the devices and to receive packets that have been forwarded to the controller. Proactive flows are established by the application; typically, the application will set these flows when the application starts up, and the flows will persist until some configuration change is made. This kind of proactive flow is known as a static flow. Another kind of proactive flow is where the controller decides to modify a flow based on the traffic load currently being driven through a network device.

In addition to flows defined proactively by the application, some flows are defined in response to a Packet forwarded to the controller. Upon receipt of incoming packets that have been forwarded to the controller, the SDN application will instruct the controller as to how to respond to the packet and, if appropriate, will establish new flows on the device in order to allow that device to respond locally the next time it sees a packet belonging to that flow. Such flows are called reactive flows. In this way, it is now possible to write software applications that implement forwarding, routing, overlay, multipath, and access control functions, among others.

There are also reactive flows that are defined or modified as a result of stimuli from sources other than packets from the controller. For example, the controller can insert flows reactively in response to other data sources such as intrusion detection systems (IDS) or the NetFlow traffic analyser fig. (2.1.2) depicts the OpenFlow protocol as the means of communication between the controller and the device. Though OpenFlow is the defined standard for such communication in Open SDN

2.1.2. SDN DEVICES

An SDN device is composed of an API for communication with the controller, an abstraction layer, and a packet-processing function. In the case of a virtual switch, this packet-processing function is packet-processing software, and in the case of a physical switch, the packetprocessing function is embodied in the hardware for packet-processing logic, The packet-processing logic consists of the mechanisms to take actions based on the results of evaluating incoming packets and finding the highest-priority match. When a match is found, the incoming packet is processed locally unless it is explicitly forwarded to the controller. When no match is found, the packet may be copied to the controller for further processing. This process is also referred to as the controller consuming the packet. In the case of a hardware switch, these mechanisms are implemented by the specialized hardware In the case of a software switch (virtual switches), these same functions are mirrored by software. Since the case of the software switch is somewhat simpler than the hardware switch.

2.1.3. FLOW TABLES

Flow tables consist of a number of prioritized flow entries, each of which typically consists of two components: match fields and actions. Match fields are used to compare against incoming packets. An incoming packet is compared against the match fields in priority order, and the first complete match is selected. Actions are the instructions that the network device should perform if an incoming packet matches the match fields specified for that flow entry. Match fields can have wildcards for fields that are not relevant to a particular match. For example, when matching packets based just on IP address or subnet, all other fields would be wild carded. Similarly, if matching on only MAC address or UDP/TCP port, the other fields are irrelevant, and consequently those fields are wild carded. Depending on the application needs, all fields may be important, in which case there would be no wildcards. The flow table and flow entry constructs allow the SDN application developer to have a wide range of possibilities for matching packets and taking appropriate actions.

2.1.4. OPENFLOW

OpenFlow is a protocol that allows a server to tell network switches where to send packets. In a conventional network, each switch has proprietary software that tells it what to do. With OpenFlow, the packet-moving decisions are centralized, so that the network can be programmed independently of the individual switches and data center gear.

In a conventional switch, packet forwarding (the data path) and high-level routing (the control path) occur on the same device. An OpenFlow switch separates the data path from the control path. The data path portion resides on the switch itself; a separate controller makes high-level routing decisions. The switch and controller communicate by means of the OpenFlow protocol

.

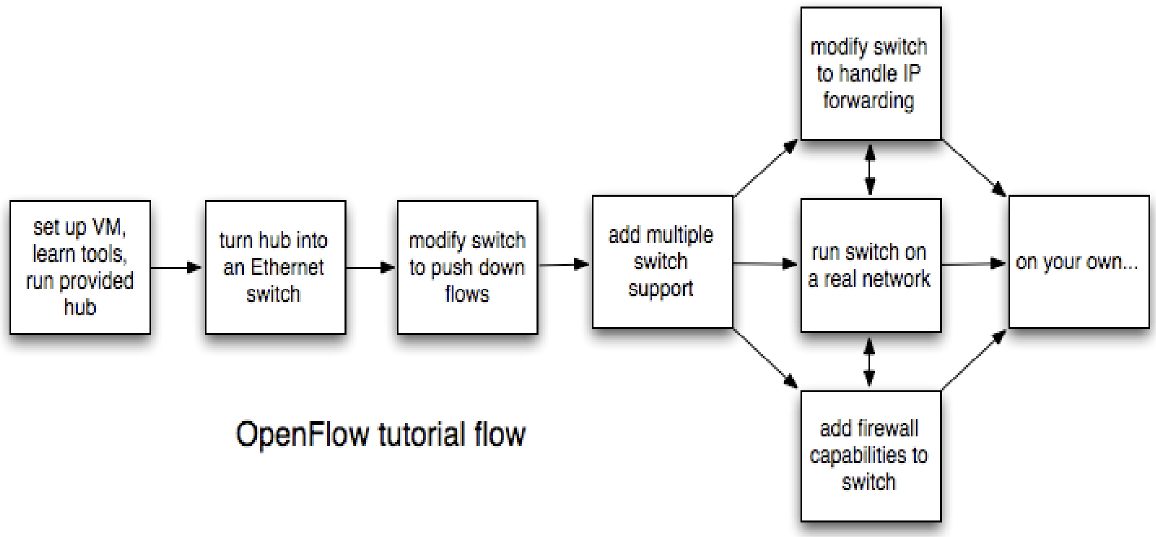

set up VM, learn tools, run provided hub
turn hub into an Ethernet switch
modify switch to push down flows
add multiple switch support
modify switch to handle IP forwarding
run switch on a real network
add firewall capabilities to switch
on your own...
OpenFlow tutorial flow

Figure 2.1.4.1 OpenFlow

This methodology, known as software-defined networking (SDN), allows for more effective use of network resources than is possible with traditional networks. OpenFlow has gained favour in applications such as VM (virtual machine) mobility, mission-critical networks, and next generation IP-based mobile networks.

Several established companies including IBM, Google, and HP have either fully utilized, or announced their intention to support, the OpenFlow standard. Big Switch Networks, an SDN firm headquartered in Palo Alto, California, has implemented OpenFlow networks that run on top of traditional networks, making it possible to place virtual machines anywhere in a data center to reclaim stranded computing capacity. By early 2012, Google's internal network ran entirely on OpenFlow.

Working of OpenFlow:

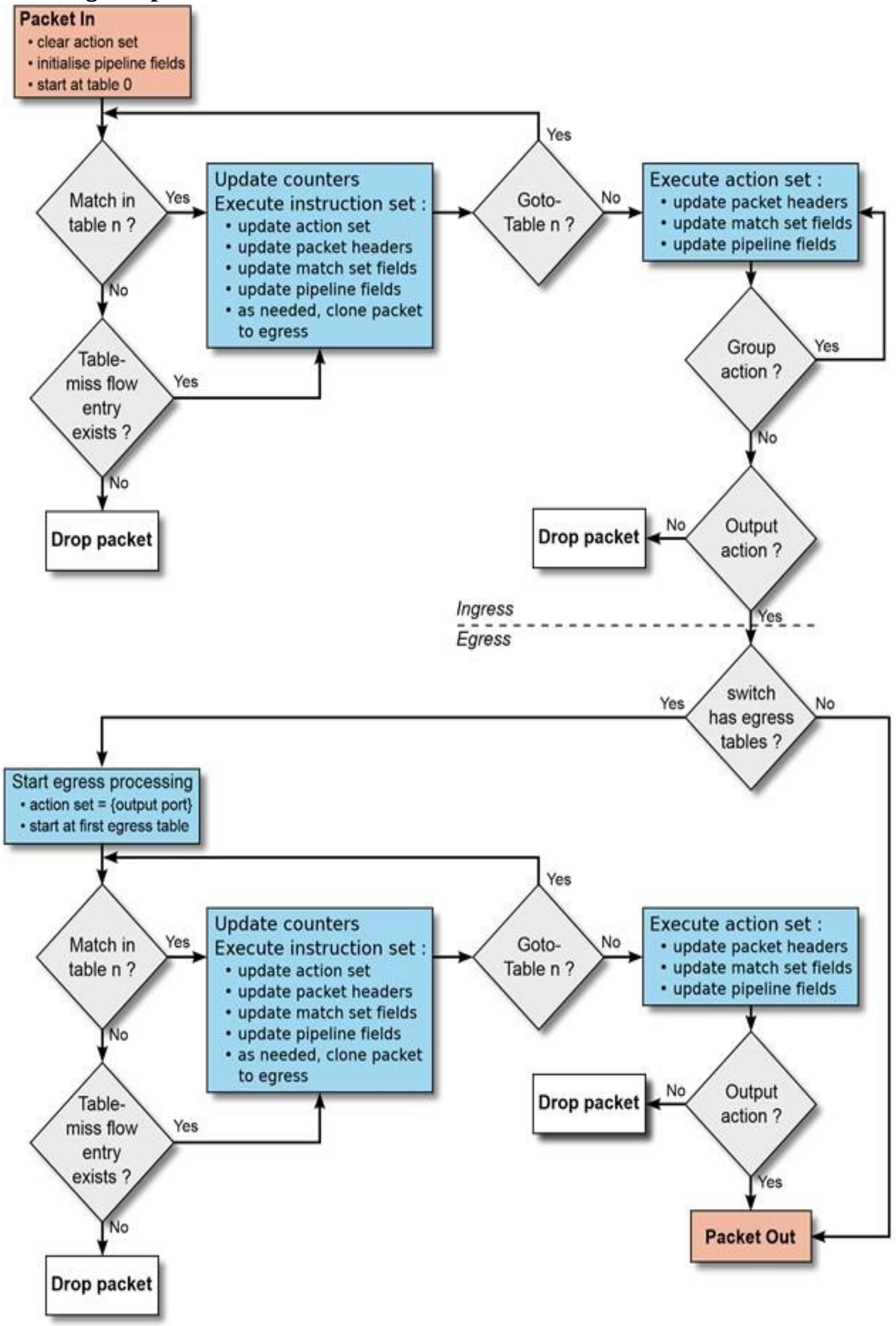


Figure 2.1.4.2 Flow entry of open flow.

2.1.5. Secure Channel

A brief look at the specification of Openflow shows the single point where the forwarding plane and control plane are connected. It is the secure channel between the controller and the switch. If the connection to the controller is lost, a pure Openflow switch will not be able to deal with unknown incoming packets. The secure channel is the lifeline of all Openflow switches in SDN. It is a TLS or TCP connection established between the controller and the switch. If the connection is lost, the switch will try to connect to a backup controller if there is one. This is the “fail secure mode” and all packets to the controller will be dropped. In an Openflow switch, all new packets will be processed in the switch and will not be sent to the controller. If the switch is capable of working both in SDN and in none SDN then, it is called Hybrid switch. In this case, the switch will not follow the Openflow protocol and the network loses its SDN architecture. The Openflow specification shows that without a controller in the network, we are dealing with a non-SDN network with no central control or separation of forwarding and control plane. This section shows the importance of detecting any threat that can make the controller unreachable.

2.2. MOTIVATION

The increasing number of attacks and the effects of these happenings invoked my interest in this subject. This unresolved issue is actively present in the IT world for nearly a decade and there has never been an ultimate solution for this. The magnitude of damage caused by DDoS pushed me to learn more about this topic and urged me to do my contribution. These attacks have got businesses down, crippled the economy of a nation and even changed government. Experts also predict that the future wars are going to be with IP packets as missiles since they are capable of bringing down a nation.

The attack which was carried out in Burma had kept the nation out of internet for several months. The traffic sent were unstoppable ranging from 10 to 15 Gbps which was several folds more than what nation's network could withstand. The whole nation was devoid if internet and the ecommerce industry came to a standstill condition. Nevertheless, there are effective solutions that are suggested in order to survive these attacks even though complete removal is not possible. Allocating extra bandwidth, tracing back the attacker, identifying and stopping the fake packets are few of the general suggestions widespread amongst experts. But the exact solution varies with the severity of the attack and the value of data that the company is trying to protect. Thus the ultimate motivation a rose with a desire of stopping these attacks that could lead to safe and secure IT world. This dissertation report gives a baseline to DDoS and helps a novice technician to understand the subject better.

2.3. PROPOSED SYSTEM

2.3.1. DDOS DETECTION USING ENTROPY

Entropy measures randomness. The main reason entropy is used for DDoS detection is its ability to measure randomness in the packets that are coming to a network. The higher the randomness the higher is the entropy and vice versa. There are two essential components to DDoS detection using entropy:

i. Window size
ii. Threshold

Window size is either based on a time period or a number of packets. Entropy is calculated within this window to measure uncertainty in the coming packets. To detect an attack, a threshold is needed. If the calculated entropy passes a threshold or is below it, depending on the scheme, an attack is detected. Let n be the number of packet in a window and p is the probability of each element in the window then, entropy (H) will be calculated as:

Entropy represents packet headers as independent information symbols with unique probability of occurrence. By selecting a window of some number, 10,000 for instance, and moving the window forward, a pattern will emerge with probabilities for each type of packet header. Drastic changes in the bins of each header that deviates from the average bin limits will alert the system of anomalies.

$$H = -\sum_{i=1}^{n} p_i \log p_i \qquad (1)$$

Entropy is a, fairly, common method for DDoS detection. Qin et al. [8] propose a method with a window of 0.1 seconds and three levels of threshold. This method is concerned with avoiding false positives and false negatives in the network. However, as the authors themselves mention, the method is time consuming and uses more resources. This method also uses a time period window. For the threshold, the authors ran several datasets to find a suitable experimental threshold and it is a multiple of the standard deviation of the entropy values. In this method, the false negatives are higher than other methods and false positives are lower. No percentage of accuracy is indicated. There is also no mention of resources used for fast computation.

Oshima et al. [10] propose a short-term statistics detection method based on entropy computation. “Short-term” here refers to calculating entropy in small size windows. The study proposes a window size of 50 packets for gathering statistics. In this method, different window sizes were tested for optimal entropy measurement and a size of 50 was the lowest size that effectively detected attacks. Instead of a threshold, the authors used a one-sided test of significance to confirm whether an attack was in progress or not. In the DDoS detection case, the rate of attack traffic was increased by increments while the normal traffic rate was kept constant. The method proved to be effective the most when the attack traffic was 75% or higher than the normal traffic.

2.3.2. Utilizing SDN Capabilities

For every new incoming connection, the controller will install a flow in the switch so that the rest of the incoming packets will be directed to the destination without further processing. Hence, any time a packet is seen in the controller, it is new. The other known fact about new packets coming to the controller is that the destination host is within the network of the controller. This assumption is based on the fact that one of the hosts or a subnet of hosts in the network is being attacked. The network consists of the switches and hosts that are connected to it. Knowing that the packet is new and that the destination is in the network, the level of randomness can be quantified by calculating the entropy based on a window size. In this case, maximum entropy occurs when each packet is send by exactly one source. On the other side, minimum entropy occurs when all the packets in a window are send by single source. For instance, if the window has 64 elements and all elements appear only once, then based on Eq. (1), the entropy will be 1.80. If one element appears 10 times, the entropy will be 1.64. This property of the entropy will be used for calculating the randomness in the SDN controller. The central view of the controller over the network gives us the opportunity to evaluate the rate of incoming new packets to the controller and decide whether an attack is in progress or not.

Being able to quantify randomness and have minimum and maximum based on the entropy makes it a suitable method for DDoS detection is SDN. Using entropy, it is possible to see its value drop when a large number of packets are attacking one host or a subnet of hosts.

2.3.3. Statistics Collection for Entropy

One of the functions of the controller is collecting statistics from the switch tables. The controller monitors the existing flows and if there is an inactive flow for a period of time, it will remove that flow. In this paper, we will make use of that property and add another set of statistics collection to the controller. Since the approximate number of hosts in the network is known, we added the destination IP address of the new incoming packets to be collected into windows of size 50. The entropy of each window is calculated and compared to an experimental threshold. If the entropy islower than the threshold, an attack is detected. These functions are small (in terms of the amount of code added) and transparent to the other functionalities of the controller. These two properties make the solution applicable to different controllers with minimal changes.

2.3.4. Window Size

The window size should be set to be smaller or equal to the number of hosts in order to provide accurate calculations. In this project, we set the window size at 50. The main reason for choosing 50 is the limited number of incoming new connection to each host in the network. In SDN, once a connection is established, the packets will not pass through the controller unless there is a new request. A second reason is the fact that a limited number of switches and hosts can be connected to each controller. Finally, a third reason for choosing this size is the amount of computation that is done for each window. A list of 50 values can be computed much faster than 500 and an attack in a 50-packet window is detected earlier. We also tested the entropy with three other window sizes and measured the CPU and memory usage. There is no difference in memory usage but the CPU usage slightly increases with respect to the window size. Considering the limited resources of the controller, this window size is ideal for networks with one controller and few hundred hosts.

2.3.5. Attack detection

To detect an attack in the controller, we monitor the destination IP address of the incoming packets. A function was added to the controller to create a hash table of the incoming packets.

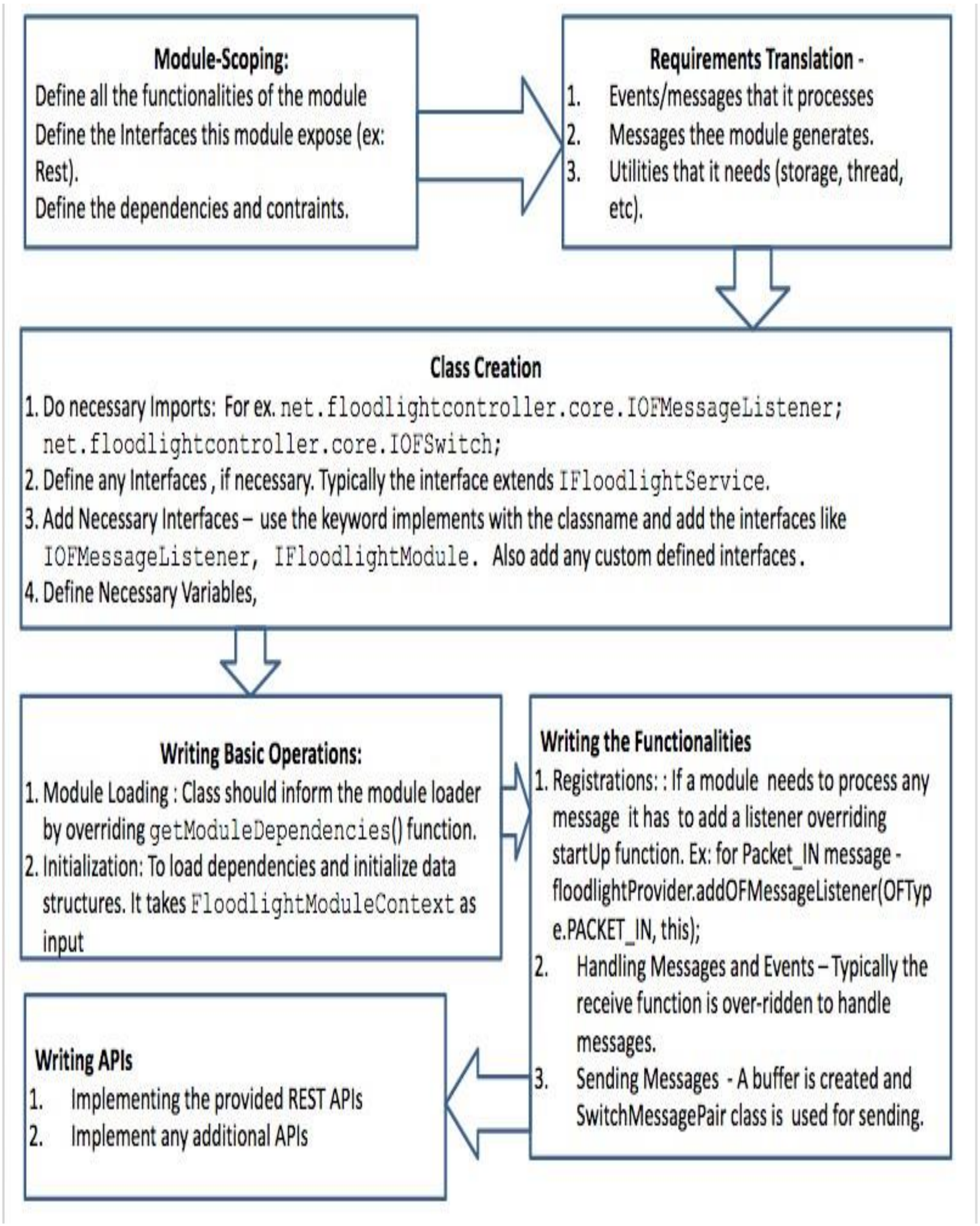
$$W = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots\} \tag{2}$$

$$p_i = \frac{x_i}{n} \tag{3}$$

If an IP address is new in the table, it will be added with count one. If there is an instance of it in the table, its count will be incremented. After 50 packets, the entropy of the window will be calculated. Eq. (2) shows the hash table where x is the destination IP address and y is the number of times it appeared. To compute the entropy as shown in Eq. (1), we use equations (2) and (3) where W is the window and p is the probability of each IP address.

When each IP address appears only once, the entropy will be at its maximum. If an attack is directed towards a host, a large number of packets will be directed to it. These packets will fill most of the window and reduce the number of unique IPs in the window, which in turn, reduces entropy. We made use of this fact and set an experimental threshold. If the entropy drops below this threshold and that five consecutive windows have lower than threshold entropy, then an attack is in progress. Detection within 5 entropy periods is 250 packets in the attack, which gives the network an early alert of the attack. We tested with different values between one and five consecutive periods and, five has the lowest false positive for early detection. The other advantage of having five windows is the possibility of losing a switch or a broken link, which will cut off some hosts and reduce the number of new packets into the controller. This will cause a drop in the entropy and a false positive. Five windows can give the network admin enough time to act.

2.4. MODULES


Module-Scoping:
Define all the functionalities of the module
Define the Interfaces this module expose (ex: Rest).
Define the dependencies and contraints.

Requirements Translation -
1. Events/messages that it processes
2. Messages thee module generates.
3. Utilities that it needs (storage, thread, etc).

Class Creation
1. Do necessary Imports: For ex. `net.floodlightcontroller.core.IOFMessageListener;` `net.floodlightcontroller.core.IOFSwitch;`
2. Define any Interfaces , if necessary. Typically the interface extends `IFloodlightService`.
3. Add Necessary Interfaces – use the keyword implements with the classname and add the interfaces like `IOFMessageListener, IFloodlightModule.` Also add any custom defined interfaces.
4. Define Necessary Variables,

Writing Basic Operations:
1. Module Loading : Class should inform the module loader by overriding `getModuleDependencies()` function.
2. Initialization: To load dependencies and initialize data structures. It takes `FloodlightModuleContext` as input

Writing the Functionalities
1. Registrations: : If a module needs to process any message it has to add a listener overriding startUp function. Ex: for Packet_IN message - floodlightProvider.addOFMessageListener(OFType.PACKET_IN, this);
2. Handling Messages and Events – Typically the receive function is over-ridden to handle messages.
3. Sending Messages - A buffer is created and SwitchMessagePair class is used for sending.

Writing APIs
1. Implementing the provided REST APIs
2. Implement any additional APIs

Figure 2.5.2 Flow chart of Module

CHAPTER 3

3. DESIGN

3.1. STEPS TO DOWNLOAD AND CONFIGURE VMWARE WITH MININET

Step1. Download VMware Workstation11 for Linux 64 bit

Step2. It will download some VMware. Bundle file

Step3. Now open the terminal

Step4. Go to the downloads folder

Step5. Type ls, and find the **.bundle** file.

Step6. Now type the following command “Chmod +x VMware. Bundle. /name”

Step7. Now VMware will be installed

Step8. Download Mininet

Step9. Restart the system

Step10. Click on VMware player icon

Step11. Click on open a virtual machine and browse to the mininet image

Step12. Your VMware window look like this as shown in figure and click on run icon

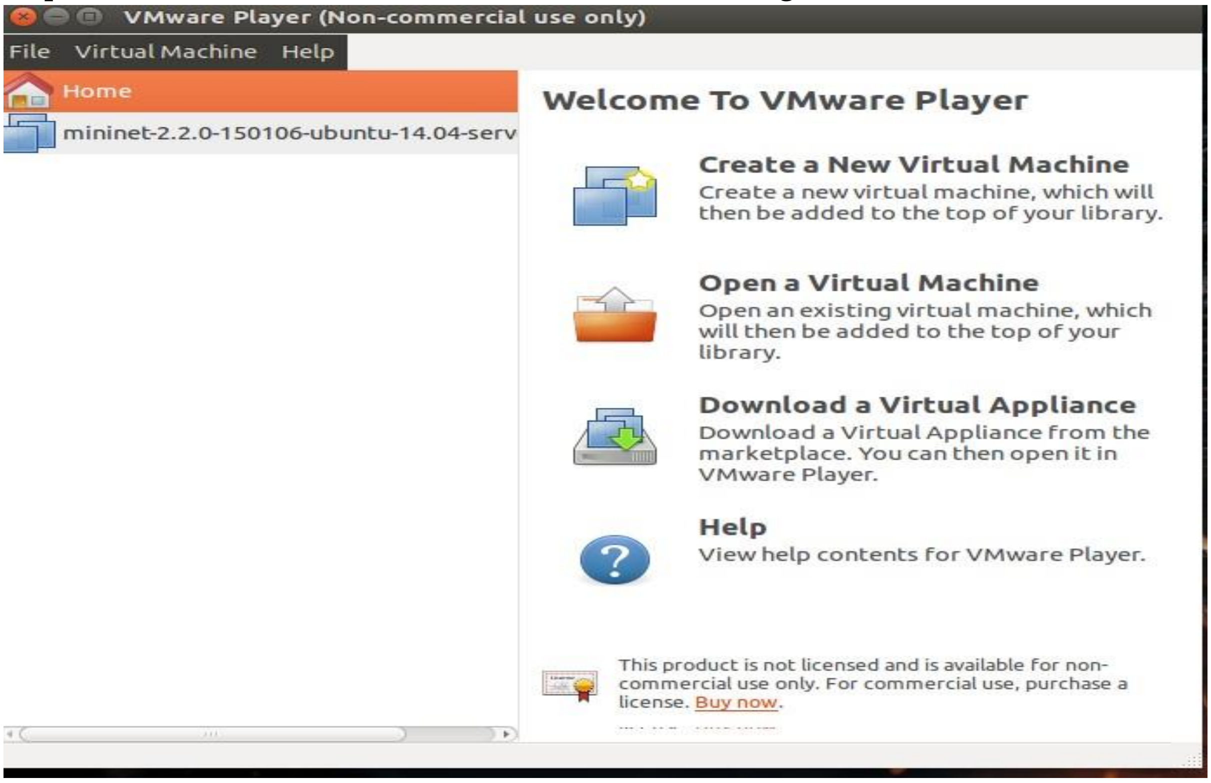
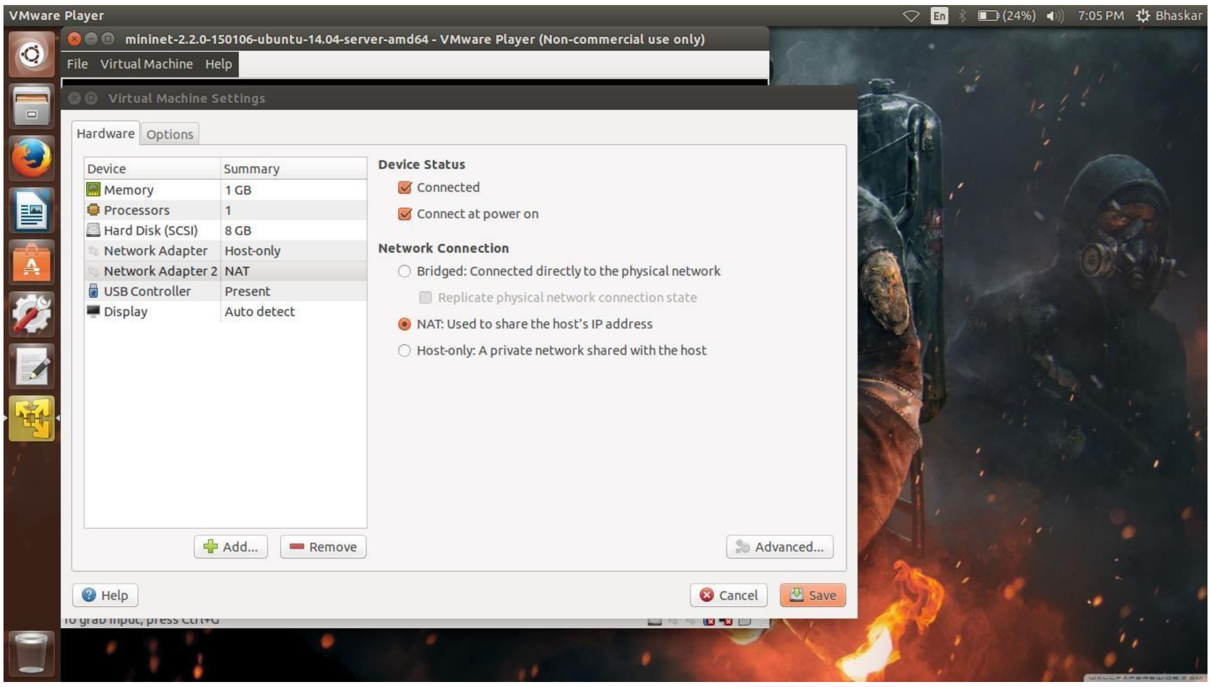

VMware Player (Non-commercial use only)
File
Virtual Machine
Help
Home
mininet-2.2.0-150106-ubuntu-14.04-serv
Welcome To VMware Player
Create a New Virtual Machine
Create a new virtual machine, which will then be added to the top of your library.
Open a Virtual Machine
Open an existing virtual machine, which will then be added to the top of your library.
Download a Virtual Appliance
Download a Virtual Appliance from the marketplace. You can then open it in VMware Player.
Help
View help contents for VMware Player.
This product is not licensed and is available for non-commercial use only. For commercial use, purchase a license. Buy now.

Figure 3.1.1 VMware Page

Step13. Click on the VMware Settings at the top left corner and add one NetworkAdapter2


VMware Player
mininet-2.2.0-150106-ubuntu-14.04-server-amd64 - VMware Player (Non-commercial use only)
File Virtual Machine Help
Virtual Machine Settings
Hardware Options
Add... Remove Advanced...
Help Cancel Save

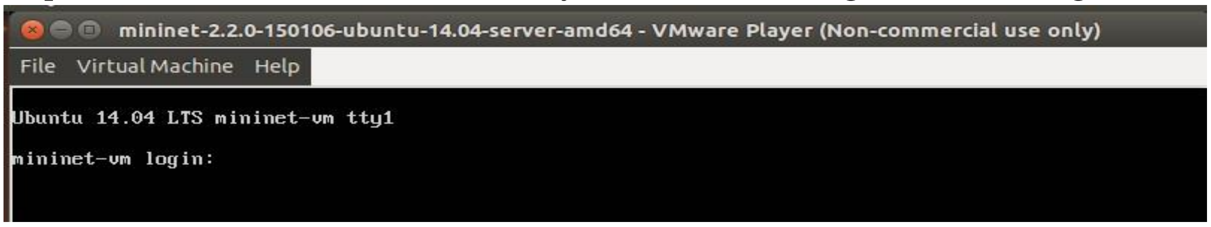
Device	Summary
Memory	1 GB
Processors	1
Hard Disk (SCSI)	8 GB
Network Adapter	Host-only
Network Adapter 2	NAT
USB Controller	Present
Display	Auto detect

Device Status
[✓] Connected
[✓] Connect at power on

Network Connection
○ Bridged: Connected directly to the physical network
☐ Replicate physical network connection state
● NAT: Used to share the host's IP address
○ Host-only: A private network shared with the host

Figure 3.1.2 VMware Settings

Step14. Our VMware with mininet is ready and it will ask for login as shown in figure

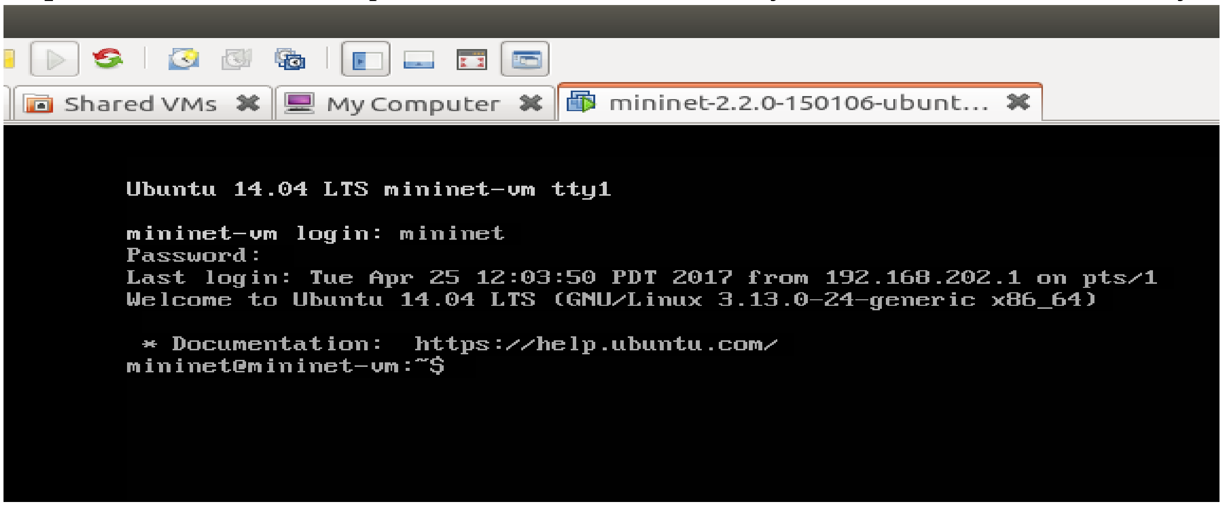

mininet-2.2.0-150106-ubuntu-14.04-server-amd64 - VMware Player (Non-commercial use only)
File Virtual Machine Help

```
Ubuntu 14.04 LTS mininet-vm tty1

mininet-vm login:
```

Figure 3.1.3 VMware Login Page

Step15. The username and password both are mininet and your VMware machine are ready



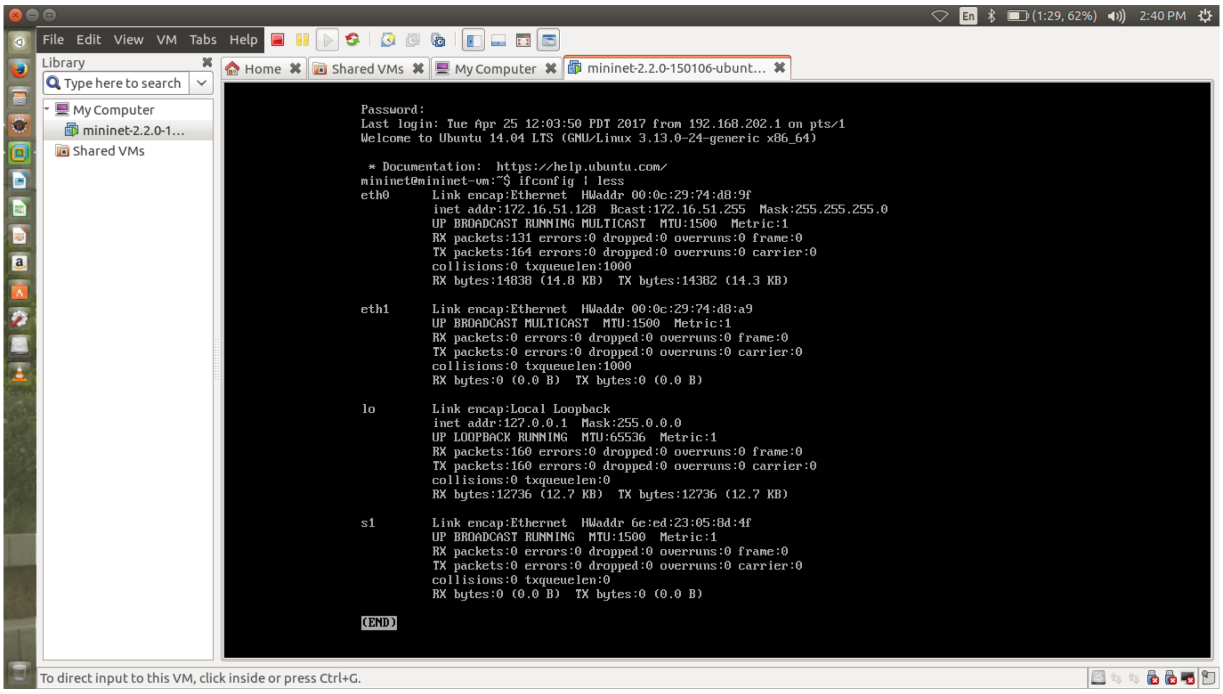
```
Ubuntu 14.04 LTS mininet-vm tty1

mininet-vm login: mininet
Password:
Last login: Tue Apr 25 12:03:50 PDT 2017 from 192.168.202.1 on pts/1
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
mininet@mininet-vm:~$
```

Figure 3.1.4 VMware Login

Step16. Open terminal and type ipconfig command and you will get VMware ip as shown in this figure 3.1.5 below


File Edit View VM Tabs Help
Library
Type here to search
My Computer
mininet-2.2.0-1...
Shared VMs
Home
Shared VMs
My Computer
mininet-2.2.0-150106-ubunt...
(1:29, 62%)
2:40 PM
To direct input to this VM, click inside or press Ctrl+G.

```
Password:
Last login: Tue Apr 25 12:03:50 PDT 2017 from 192.168.202.1 on pts/1
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
mininet@mininet-vm:~$ ifconfig | less
eth0      Link encap:Ethernet  HWaddr 00:0c:29:74:d8:9f
          inet addr:172.16.51.128  Bcast:172.16.51.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:131 errors:0 dropped:0 overruns:0 frame:0
          TX packets:164 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14838 (14.8 KB)  TX bytes:14382 (14.3 KB)

eth1      Link encap:Ethernet  HWaddr 00:0c:29:74:d8:a9
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:160 errors:0 dropped:0 overruns:0 frame:0
          TX packets:160 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:12736 (12.7 KB)  TX bytes:12736 (12.7 KB)

s1        Link encap:Ethernet  HWaddr 6e:ed:23:05:8d:4f
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

(END)
```

Fig 3.1.5 Ifconfig Page

Step17. Now take ssh login by typing following command “ssh mininet@172.16.32.128”

Step18: Install and configure scapy by following commands:

- install package of scapy.... scapy-latestversion.tar.gz
- place it at home directory
- type ls to see it
- type tar -xzvf file.tar.gz
- type cd scapy-2.3.2
- type ls
- . /run_scapy

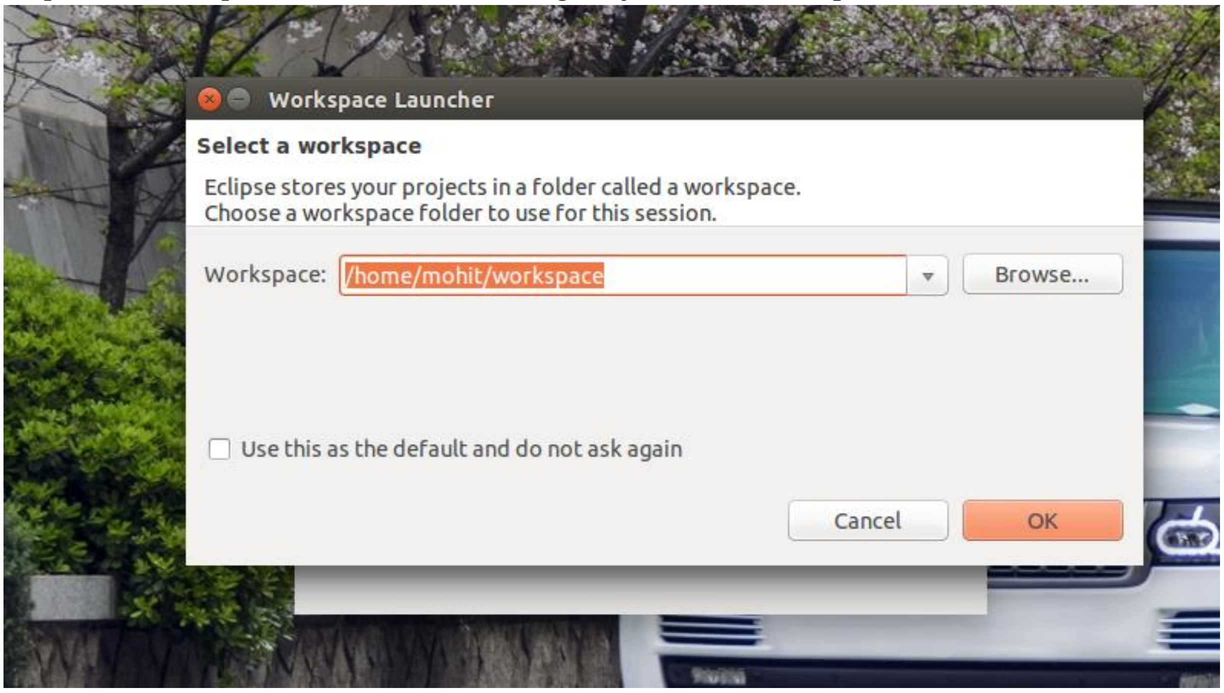
3.2. INSTALL AND CONFIGURE FLOODLIGHT CONTROLLER

Follow the below commands;

- sudo apt-get install build-essential default-jdk ant python-dev eclipse
- git clone git://github.com/floodlight/floodlight.git
- cd floodlight
- jar -jar target/floodlight.jar
- ant eclipse
- eclipse&

Open eclipse and create a new workspace (inside Eclipse);

Step1. File -> Import -> General -> Existing Projects into Workspace. Then click “Next”.



Workspace Launcher

Select a workspace

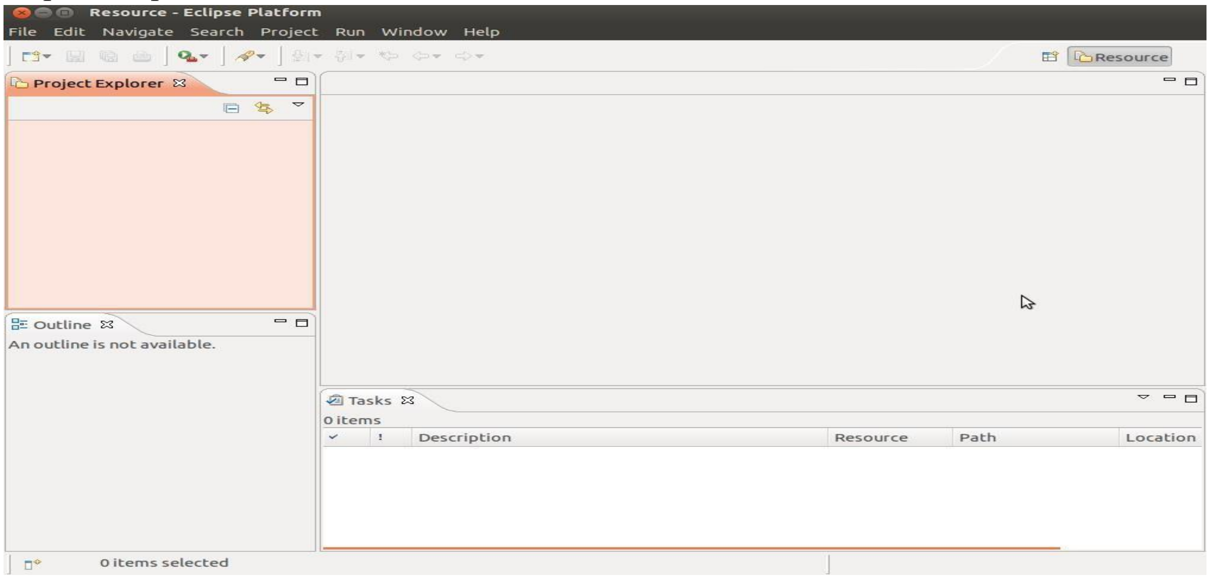
Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: /home/mohit/workspace

Use this as the default and do not ask again

Figure 3.2.1 Eclipse Workstation

Step2. Eclipse shows “welcome”, close welcome tab.



Resource - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Resource
Project Explorer
Outline
Tasks

An outline is not available.

0 items

✓	!	Description	Resource	Path	Location

0 items selected

Figure 3.2.2 Project Explorer.

Step3. “File”- “import” – “Existing Projects into Workspace”.

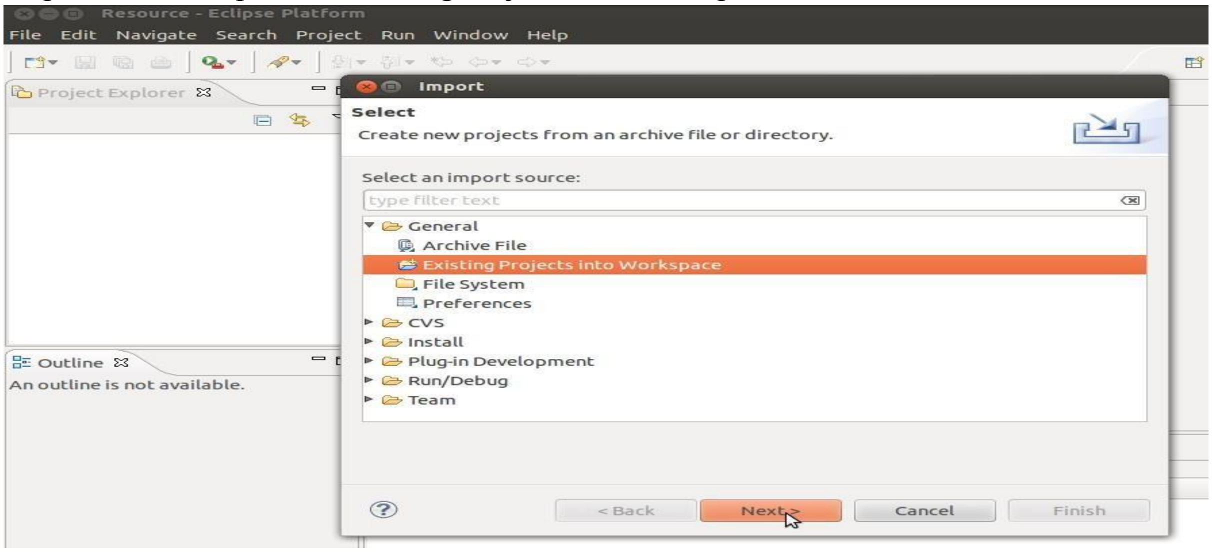

Resource - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Project Explorer
Outline
An outline is not available.
Import
Select
Create new projects from an archive file or directory.
Select an import source:
type filter text
General
Archive File
Existing Projects into Workspace
File System
Preferences
CVS
Install
Plug-in Development
Run/Debug
Team
< Back
Next >
Cancel
Finish

Figure 3.2.3 Import

Step4. From “Select root directory” click “Browse”. Select the parent directory where you placed floodlight earlier.

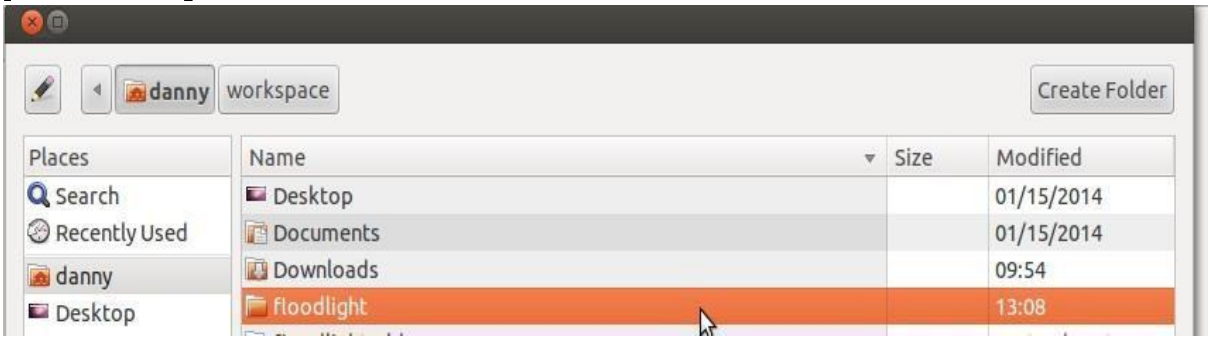


Figure 3.2.4 Import Floodlight

Step5. Check the box for “Floodlight”. No other Projects should be present and none should be selected and Click Finish.

Step6. Then, we see “floodlight” in the “Project Explorer”.

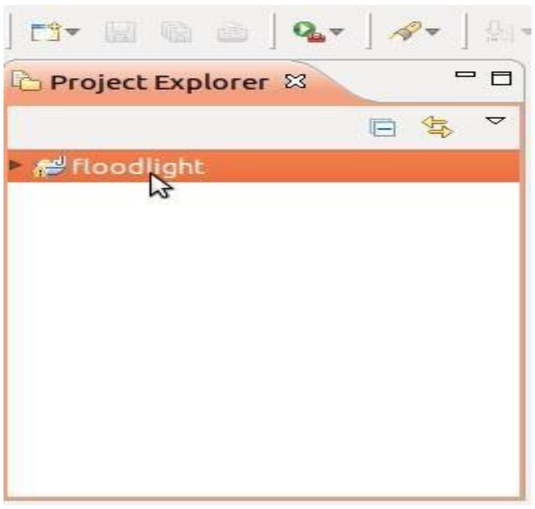
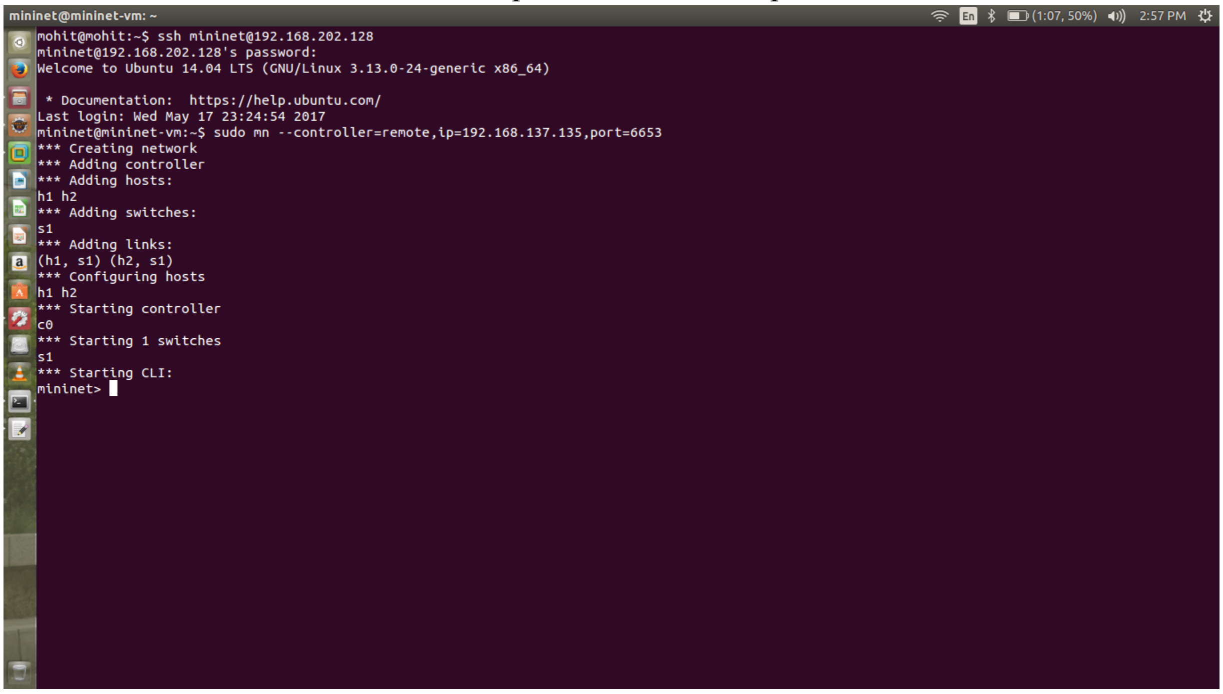


Figure 3.2.5 Floodlight project

Step7. Create the FloodlightLaunch target (inside Eclipse);

- Click Run->Run Configurations
- Right Click Java Application->New
- For Name Use Floodlight Launch
- For Project use Floodlight
- For Main use net. floodlightcontroller. core. Main
- Click Apply

Type following command in your ssh logged terminal
"sudo mn –controller = remote,ip=192.168.238.102,port=6633"



```
mininet@mininet-vm: ~
mohit@mohit:~$ ssh mininet@192.168.202.128
mininet@192.168.202.128's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Wed May 17 23:24:54 2017
mininet@mininet-vm:~$ sudo mn --controller=remote,ip=192.168.137.135,port=6653
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Figure 3.2.7 Floodlight Connection

CHAPTER 4

4. IMPLEMENTATION

4.1. FLOODLIGHT MODULE ONE

4.1.1. Prerequisites

We are going to create a bundle that will watch for new MAC addresses that have not been seen before, and log the MAC and switch they were seen on.

Successfully completed the floodlight installation, including setting up Eclipse Mininet

installed and running, or a physical OpenFlow switch

4.1.2. Creating the Listener

Add Class in Eclipse:

- Expand the "floodlight" item in the Package Explorer and find the "src/main/java" folder.
- Right-click on the "src/main/java" folder and choose "New/Class".
- Enter "net.floodlightcontroller.ddos" in the "Package" box. Enter "DDOS" in the "Name" box.
- Next to the "Interfaces" box, choose "Add...”
- Add the "IOFMessageListener" and the "IFloodlightModule", click "OK". Click "Finish" in the dialog.

4.1.3. Register the Module

We're almost done, now we just need to tell Floodlight to load the module on start-up. First we have to tell the loader that the module exists. This is done by adding the fully qualified module name on its own line in src/main/resources/META-INF/service net.floodlight.core.module.IFloodlightModule. We open that file and append this line.

net.floodlightcontroller.ddos.DDOS

Then we tell the module to be loaded. We modify the Floodlight module configuration file to append the DDOS. The default one is src/main/resources/floodlightdefault.properties. The key is floodlight.modules and write following line **net.floodlightcontroller.ddos.DDOS**

Finally, let's run the controller by right clicking on Main.java and choose "Run As.../Java Application".

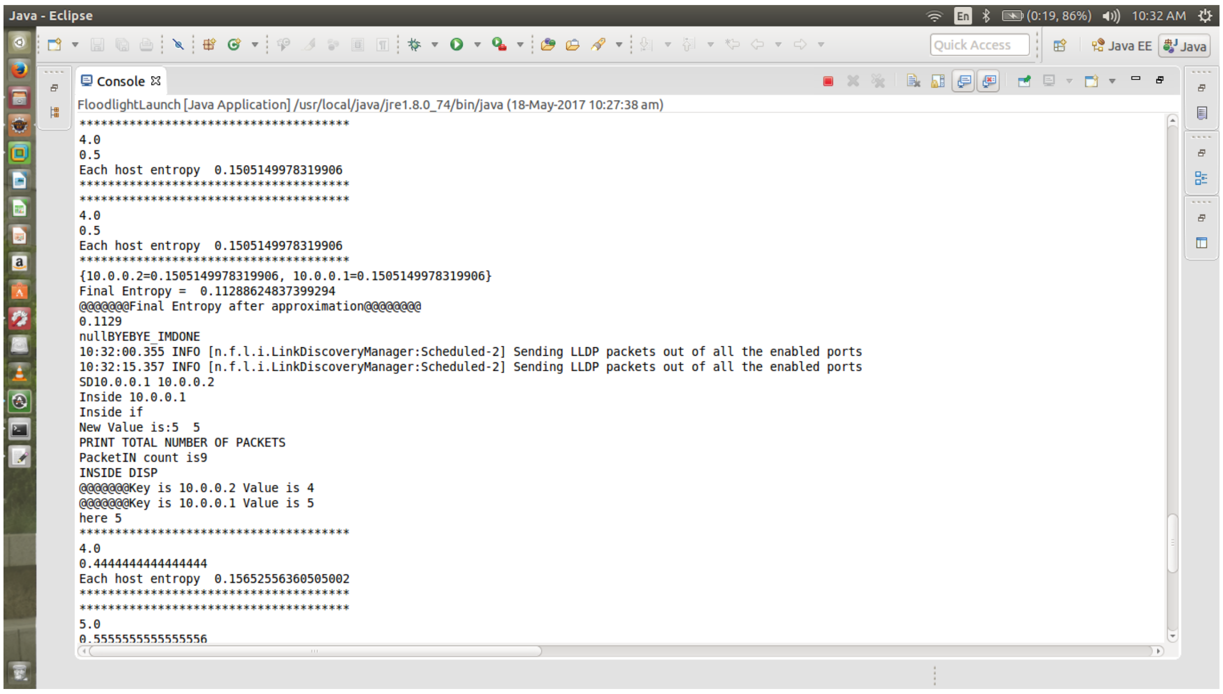
CHAPTER 5
5. OUTPUT SCREENS

Type following command in your terminal

- ssh -X mininet@172.16.32.128 //command to take ssh login
- sudo mn –top tree,3 –controller=remote, ip=192.168.43.130, port=6653 –switch=ovsk , protocols=OpenFlow13
- pingall

5.1. CASE

Output screen for the normal traffic, where each host has the same probability of number of packets sending, in this case we have total 120 packets in a window



FloodlightLaunch [Java Application] /usr/local/java/jre1.8.0_74/bin/java (18-May-2017 10:27:38 am)

```
*************************************
4.0
0.5
Each host entropy  0.1505149978319906
*************************************
*************************************
4.0
0.5
Each host entropy  0.1505149978319906
*************************************
{10.0.0.2=0.1505149978319906, 10.0.0.1=0.1505149978319906}
Final Entropy =  0.11288624837399294
@@@@@@@Final Entropy after approximation@@@@@@@@
0.1129
nullBYEBYE_IMDONE
10:32:00.355 INFO [n.f.l.i.LinkDiscoveryManager:Scheduled-2] Sending LLDP packets out of all the enabled ports
10:32:15.357 INFO [n.f.l.i.LinkDiscoveryManager:Scheduled-2] Sending LLDP packets out of all the enabled ports
SD10.0.0.1 10.0.0.2
Inside 10.0.0.1
Inside if
New Value is:5  5
PRINT TOTAL NUMBER OF PACKETS
PacketIN count is9
INSIDE DISP
@@@@@@@Key is 10.0.0.2 Value is 4
@@@@@@@Key is 10.0.0.1 Value is 5
here 5
*************************************
4.0
0.4444444444444444
Each host entropy  0.15652556360505002
*************************************
*************************************
5.0
0.5555555555555556
```

Figure 5.1.1 Output_Screen_1_NormalTraffic

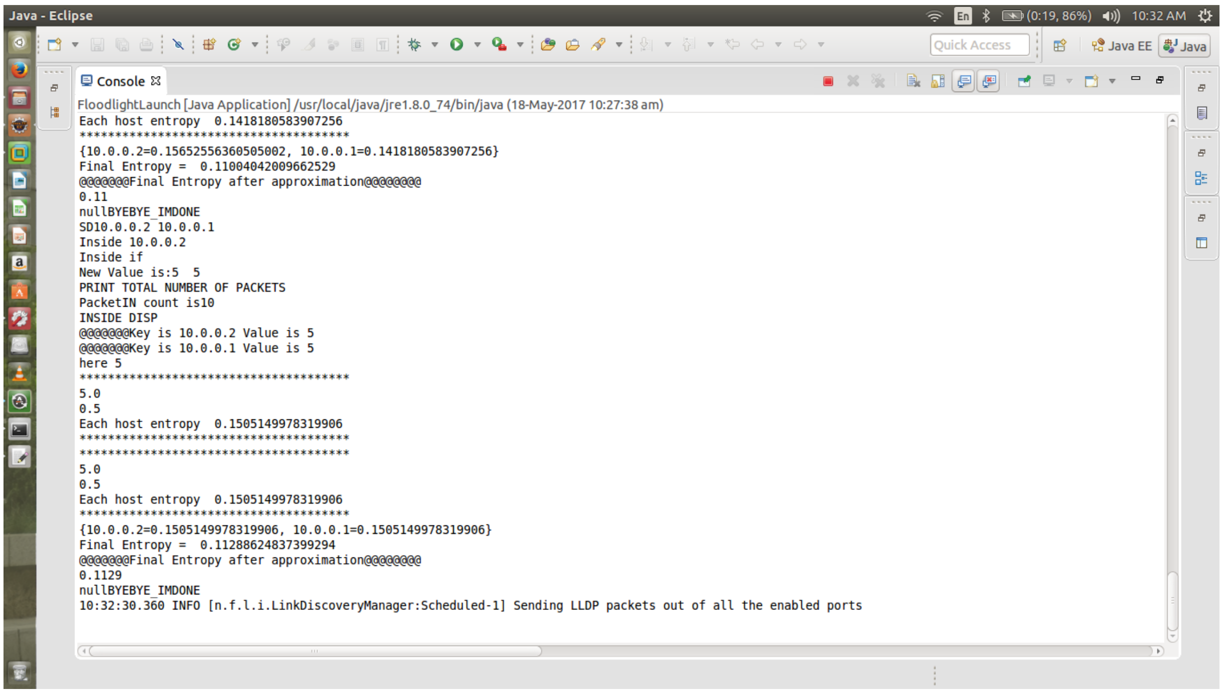
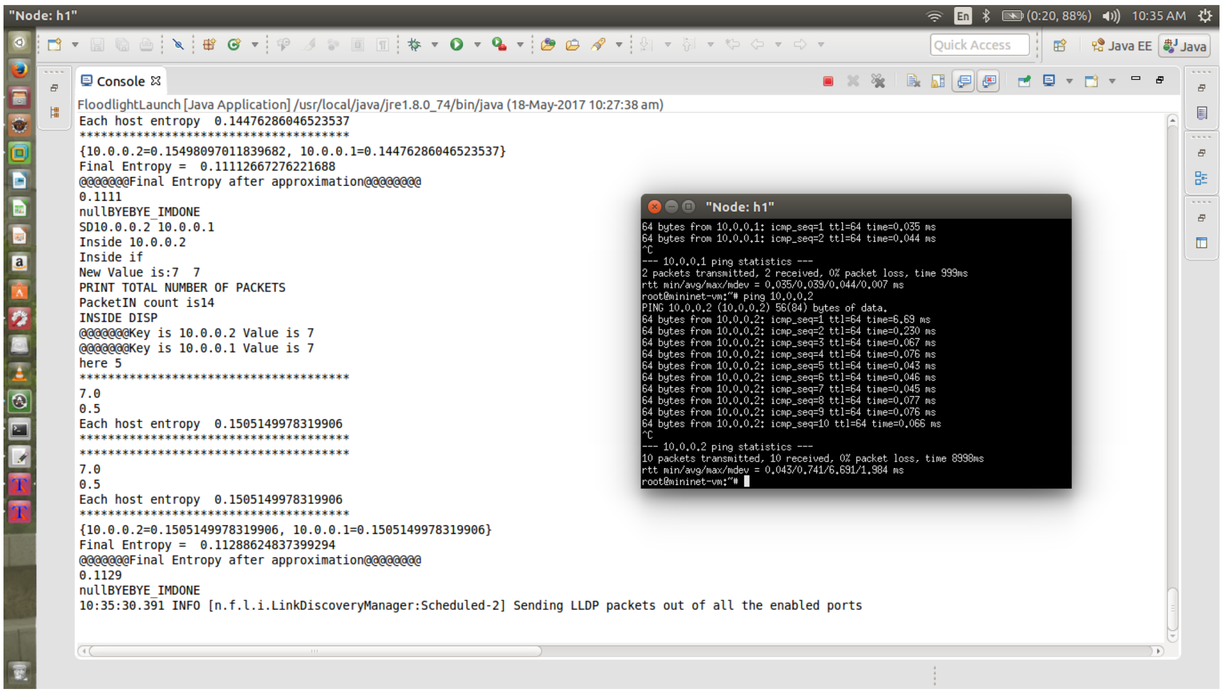

```
FloodlightLaunch [Java Application] /usr/local/java/jre1.8.0_74/bin/java (18-May-2017 10:27:38 am)
Each host entropy  0.1418180583907256
**************************************
{10.0.0.2=0.15652556360505002, 10.0.0.1=0.1418180583907256}
Final Entropy =  0.11004042009662529
@@@@@@@Final Entropy after approximation@@@@@@@
0.11
nullBYEBYE_IMDONE
SD10.0.0.2 10.0.0.1
Inside 10.0.0.2
Inside if
New Value is:5  5
PRINT TOTAL NUMBER OF PACKETS
PacketIN count is10
INSIDE DISP
@@@@@@@Key is 10.0.0.2 Value is 5
@@@@@@@Key is 10.0.0.1 Value is 5
here 5
**************************************
5.0
0.5
Each host entropy  0.1505149978319906
**************************************
**************************************
5.0
0.5
Each host entropy  0.1505149978319906
**************************************
{10.0.0.2=0.1505149978319906, 10.0.0.1=0.1505149978319906}
Final Entropy =  0.112886248373992 94
@@@@@@@Final Entropy after approximation@@@@@@@
0.1129
nullBYEBYE_IMDONE
10:32:30.360 INFO [n.f.l.i.LinkDiscoveryManager:Scheduled-1] Sending LLDP packets out of all the enabled ports
```

Figure 5.1.2 Entropy Screen Normal Traffic


```
FloodlightLaunch [Java Application] /usr/local/java/jre1.8.0_74/bin/java (18-May-2017 10:27:38 am)
Each host entropy  0.14476286046523537
**************************************
{10.0.0.2=0.15498097011839682, 10.0.0.1=0.14476286046523537}
Final Entropy =  0.11112667276221688
@@@@@@@Final Entropy after approximation@@@@@@@
0.1111
nullBYEBYE_IMDONE
SD10.0.0.2 10.0.0.1
Inside 10.0.0.2
Inside if
New Value is:7  7
PRINT TOTAL NUMBER OF PACKETS
PacketIN count is14
INSIDE DISP
@@@@@@@Key is 10.0.0.2 Value is 7
@@@@@@@Key is 10.0.0.1 Value is 7
here 5
**************************************
7.0
0.5
Each host entropy  0.1505149978319906
**************************************
**************************************
7.0
0.5
Each host entropy  0.1505149978319906
**************************************
{10.0.0.2=0.1505149978319906, 10.0.0.1=0.1505149978319906}
Final Entropy =  0.11288624837399294
@@@@@@@Final Entropy after approximation@@@@@@@
0.1129
nullBYEBYE_IMDONE
10:35:30.391 INFO [n.f.l.i.LinkDiscoveryManager:Scheduled-2] Sending LLDP packets out of all the enabled ports
```

```
"Node: h1"
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.035 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.044 ms
^C
--- 10.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.035/0.039/0.044/0.007 ms
root@mininet-vm:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=6.69 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.230 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.043 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.046 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.045 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.077 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.066 ms
^C
--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8998ms
rtt min/avg/max/mdev = 0.043/0.741/6.691/1.984 ms
root@mininet-vm:~#
```

Figure 5.1.3 Sending packets from host 1 to host 2

CHAPTER 6

FUTURE WORK & LIMITATIONS

limitation that our method has is the detection of attacks when the entire network is being targeted by DDoS. When malicious packets are targeting every host, entropy might not change by a large margin. Detecting such attacks will be an addition to this research. Having addressed the detection in one controller network, two more tasks to be done are: i) Detection of attack in a multi-controller SDN structure ii) Mitigation of the attack.

In SDN, networks are connected to controllers and, several controllers might be connected to each other. Detecting an attack in one of them could show the source of the attack and make discovery of the source much easier. This method requires an inter-controller communication that sends the threat alert to all the controllers. Adding this communication process to SDN will be an extension to the current work and a topic for future work.

Mitigation of DDoS in SDN is the next future work for this research. The first step of mitigation will be detection of the source or sources of the attack. Adding more statistics collection to the controller will enable it to monitor the flow rate at the switch level where attack flows are directed to the controller. Then, more elaborate techniques can be used to pinpoint the malicious hosts. This is a very interesting future work that can be a baseline for any detection scheme in SDN structure.

CHAPTER 7

CONCLUSION

Protecting the operating system of SDN (i.e. the controller) by detecting DDoS attacks was the centre of this research. The challenge in detecting any threats to the controller is early detection. Although the term "early" can be used loosely in detecting an attack, we quantified the early detection to the first 250 packets of traffic as minimum and 500 packets maximum.

This solution is not only efficient in detection, it has minimal code addition to the controller program and does not increase CPU load in either normal or attack condition. There are different methods for detecting attacks and each method is used differently.

In this research, we focused on a solution that works particularly well for SDN, based on its specifications, points of strength and limitations. We made use of the fact that SDN specification dictates the forwarding of new packets to the controller. We took into account the abilities of the controller and its broad view of the whole network and used that for adding entropy statistics collection. Finally, understanding the importance of keeping the controller connected to the network at all times, we came up with a solution to detect any threat at its very beginning. Entropy has been used in DDoS detection in non-SDN network but, to the best of our knowledge, it has not been used in SDN and this is the first solution of its kind in SDN.

CHAPTER 8

REFERENCES

[1] Software Defined Networks.http://dx.doi.org/10.1016/B978-0-12-41667500004-8

[2] Jean Tourrilhes, Puneet Sharma,Sujata Banerjee, The Evolution of SDN and OpenFlow: A standard perspective.

[3] ONF White Paper, Software-Defined Networking: The New Norm for Networks, April 13, 2012.

[4] Sridhar Rao, SDN Series Part Five: Floodlight, an OpenFlow Controller, 11 months ago.

[5] Vmware Download: http://partnerweb.vmware.com/GOSIG/Ubuntu_14_04.html

[6] Mininet Images: https://github.com/mininet/mininet/wiki/Mininet-VM-Images

[7] Floodlight Controller: http://www.projectfloodlight.org/

[8] Floodlight Controller: https://floodlight.atlassian.net/

[9] Floodlight Controller: http://www.projectfloodlight.org/blog/

[10] Wang B, Zheng Y, Lou W, et al. DDoS Attack Protection in the Era of Cloud Computing and Software-Defined Networking. In 2014 IEEE 22nd International Conference on Network Protocols (ICNP), IEEE, 2014: 624-629.

[11] Wang A, Guo Y, Hao F, et al. Scotch: Elastically Scaling up SDN Control-Plane using vSwitch based Overlay. In Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, ACM, 2014: 403414.ACM,2013,43(4): 3-14.

[12] Seyed Mohammad Mousavi and Marc St-Hilaire,"Early Detection of DDoS Attacks againstSDN Controllers", Department of Systems and Computer Engineering Carleton University, Ottawa, Canada

[13] T. Nakashima, T. Sueyoshi S. Oshima, "Early DoS/DDoS Detection Method using Short-term Statistics," International Conference on Complex, Intelligent and Software Intensive Systems, 2010, pp. 168-173.