Emulation and Analysis of Software-Defined Networks for the Detection of DDoS Attacks

Sanjana Prasad, Ashwani Prasad, Karmel Arockiasamy, Xiaohui Yuan

Emulation and Analysis of Software-defined Networks for the detection of DDoS attacks

Sanjana Prasad[1], Ashwani Prasad[2,*], Karmel Arockiasamy[3], Xiaohui Yuan[4]

[1,2,3] Vellore Institute of Technology, Chennai, Tamil Nadu 600127, India
[4] University of North Texas, Denton, TX 76203, USA

[1]sanjana.prasad2019@vitstudent.ac.in [2]ashwani.prasad2019@vitstudent.ac.in
[3]karmel.a@vit.ac.in [4]xiaohui.yuan@unt.edu

***Corresponding author:** Ashwani Prasad; Email: ashwani.prasad2019@vitstudent.ac.in

Abstract. Development in digital infrastructure have resulted in distributed denial of service attacks to become increasingly widespread and target large number of host machines in organizations, that crashes down access when requested by the user. Such attacks when deployed on a larger scale, can result in a wastage of time and resources. With increasing complexity, these attacks have become more sophisticated and complex, leading to the emergence that has led to the rise of several algorithms and tools, that uses SDN (Software-Defined Networking) networks, in the domain of network security. Through this paper, we have surveyed what is a DDoS attack, its different forms and the use of software-defined networking tools such as Mininet and HPE VAN SDN controller, to analyze some basic topologies. Four different topologies were created namely Single, Linear, Tree and Torus, and experiments were performed to analyze the flow of traffic in these topology networks. From these experiments performed, the ping statistics in terms of minimum, maximum, average and minimum deviation Round-Trip Time (RTT) in milliseconds were computed for each of the given topologies. The values obtained as ping statistics were tabulated and represented graphically. A visual analysis between number of nodes and average round-trip time for the three topologies were done graphically. We have compared and analyzed several open-source tools, on their basis of efficiency of detecting the attack. Apart from this, a study of several instances of DDoS attacks in organizations regarding the mitigations and step initiated to withstand the intensity of these attacks, has also been done.

Keywords: Distributed denial of service; software-defined networks; Mininet; OpenFlow; cyber-attacks; topology

1 Introduction

With booming technologies such as big data, cloud computing and SaaS (Software as A Service) becoming emergent, we have reached a data rich era, where organizations utilize and mine tons of data from various sources to analyze and offer a variety of services as per the requirements specified by the customers. With the usage of data mining, customer segmentation and predictive analytics having a rising trend, there is a burgeoning need for a well-defined-network infrastructure that will be able to handle and facilitate the needs and requirements of such complex technologies in order to be deployed in a turbulent manner, to ensure its smooth functioning. Software-defined networks are an interface that leverages the use of software modelling in the form of interfaces and applications to interact with the base hardware and regulate the traffic mechanism in a traditional network. By using such an organized paradigm, we are able to virtualize the network, in order to fulfil the demands of any complex and robust technologies that require a smooth, hassle-free functioning network. Though, the demands of a well efficient network infrastructure have been emphasized, there is also a pressing need, for a well-developed, robust and a fool-proof system that can successfully detect any kind of threat and malicious activity that can cause hindrance to a network.

The most common attacks that are launched on most of the services today are the Denial of Service (DoS) and the Distributed Denial of Service (DDoS). When such attacks are successfully launched, they target a particular system network that could potentially cause a system to temporarily shut down or crash, restricting access to the users, intending to avail the service [1]. This task of suspending the service is achieved by increasing the traffic at the target host, thereby jamming the network by transmitting data packets that trigger breakdown. In this manner, a successful DoS attack launched denies access to services requested by the user. Though the severity of such an attack may not be critical, it can result in hefty loss of time and money. A DDoS attack is a version of a DoS attack, that deploys multiple hosts, for flooding the network with humongous traffic. In order to achieve a voluminous scale of attack, The attacks are launched using crawlers and botnets, these botnets can be used to leverage substantial amounts of infected machines, to flood the host machines with voluminous traffic. A botnet, comprises of several devices that are connected as a network, these devices could be an IoT device or a PC etc., wherein the security is controlled by a third party. Though DDoS attack is a type of DoS attack, it is used on a large scale due to its innate features, that sets it apart. When launched, a DDoS attack, shall be executed on a massive scale, due to a vast network of infected computers or a zombie army, controlled by the attacker. This makes it difficult to detect the location of the attacking party and the target host machine, shall find it tough to detect the

incoming traffic, due to the random distribution of attacking systems [2]. The disastrous impacts and outcomes, its diversity, and several types of these attacks, have resulted in the design and evolutions of systems, that can make analysis, detection, and mitigation of these attacks on the top most priority [3].
Apart from various methodologies that include the implementation of algorithms for detecting the DDoS attacks, there have been many advancements that have leveraged the use of software-defined networks (SDNs). Many methodologies have proposed the usage of solutions with SDNs, leading to widespread deployment in the field of cyber security, as it enables the central control of the network by being programmed manually and also controls and distributes incoming traffic [4]. Such a technology enables dynamic network configuration to facilitate the network performance and promote efficient management similar to a cloud computing environment.
The main outcomes of this paper can be summarized as follows:

- Study and analyze the different varieties of DDoS attacks and hence survey the various kinds of SDN controllers, that have been experimented to mitigate the attack.
- Study and understand the topologies of virtual SDNs.
- Use of HPE VAN SDN controllers to study and experiment the topologies of virtual SDNs.
- Conduct experiments to study the topology created and analyze the runtime.

The rest of the paper is organized as follows: In Section 2, we shall survey and study the diverse types of DDoS attacks and a variety of mitigation techniques that have been deployed by organizations to handle large scale attacks and survey various topologies, tools used and the accuracy achieved in detecting the attack. Section 3 gives the details about the SDN architecture. Section 4 talks about tools used for conducting the experiments and its theoretical methods. Section 5 deals with the creation of the several network topologies. Section 6 illustrates two experiments which forms the base of this research work. Section 7 comprises of the tabulated results from the conducted experiments. Finally, Section 8 concludes this research work and gives ideas for future implementation.

2 Related Work

With the emergence of technologies, that support a well-developed infrastructure, the variety, complexity, and the diversity of the type of attacks also increases, leading to a plethora of tools and models that can handle the intensity of the

attacks. The many types of DDOS attacks are: UDP flood, ICMP flood, SYN flood, Ping of Death (POD), NTP amplification and HTTP flood. With a diverse variety in DDoS attacks, there are several mitigation tools that are on the rise, that varies with the intensity of the attack. While most of the tools are similar in terms of architecture, the mitigation tools of DDoS can be classified into IRC based DDoS tools and agent-based DDoS tools [5]. In [6], Trinoo, is an open-source tool that has been widely used. It is a bandwidth depletion attack tool, that can launch a UDP flood attack against many IP addresses [7]. Trinoo is known to support IP address spoofing. The research work in [8] talks about TFN, Tribal Flood Network, offers the attacker an option to launch bandwidth and resource depletion attacks. It involves the usage of a command line interface, that serves as a medium of interaction between the attacker and the master program, but offers no encryption between the target host and the attacker.

Common attacks than can be launched by TFN are smurf, UDP flood, TCP Syn flood etc. Further, [9] elaborates on the TFN2K based on the TFN architecture and adds an encrypting message medium between various attack components [10]. The communication between the attacker and the master program takes place using an encryption algorithm CAST-256, that is key based [11]. In [12], Mstream is a tool that uses spoofing of TCP packets and ACK flags to attack the target network. It is a simple tool with point-to-point communication, wherein communication takes place through TCP and UDP packets and the primary node is connected via a telnet and zombie. A DOS attack can be designed as an attack launched to hinder the normal functionality of offering service to the clients [13]. A DoS attack is a resultant of a one-way blockage between the host user on one end and the target system on the other end, when there is an intentional blockage of service from one end. In such an attack, there is no damage that is caused to the data, but instead there is a denial to access the requested service at a particular instant of time.

Most DOS attacks are target the bandwidth of the network, which results in high traffic amongst the network, leading to more resources being consumed, due to which the request of the user is denied upon permission. SDN is an abstract networking paradigm, where in the entire system is seen as a distributed model, where there is a distinction between the data and the control planes. The data plane comprises of physical devices such as routers and switches, that are involved in transferring the data packets, while the control plane comprises of the controller. An SDN controller is usually the brain of the network, that will be able to adapt to changes and modifications, if any in the network, and performs several computations [14]. Depending upon the approach, protocols, and many other programmable features, Several SDN controllers can be contrasted on the basis of their performance using Chench, an OpenFlow tool [15]. An initial survey was conducted in [16] featuring a limited number of controllers such as Beacon,

Maestro and NOX-MT where the only criteria for evaluation has been the performance, however these have become obsolete and have been replaced with new ones like OpenDayLight and FloodLight.

In [17], the criteria considered for the performance of SDN controllers: TLS support, GUI, RESTful API, Open-Flow Support etc. This study was done using the Analytic Hierarchy process, through an extrapolation mechanism that matches the parameter of these values to a pre-defined scale. Using this technique, five controllers have been compared and "RYU" was classified to be the most suitable as per requirements. Besides, the work presented in [18] surveyed SDN OpenFlow controllers, where in an analysis of widely used controllers such as NOX, POX and Beacon was done on the basis of their efficiency. It was therefore concluded that these controllers posed some threats when it came to security and presented mismatches with workload and beacon was one of the most productive controllers in terms of output. Lastly, [19] takes into account, the operating modes namely- Reactive and Proactive, wherein proactive presents a much greater performance than reactive, since instructions are loaded onto the switch in the initial phase of Proactive mode and the switch receives a data packet, with no rule in flow table.

As increased organizations adopt evolving technologies like big data, SaaS and cloud computing, organizations are constantly scaling up their network infrastructure, DDOS attacks are a common way of disruption by hackers. Since the attacks are easy to organize and execute, the intensity of recent attacks has become more complex with time. There have been multiple instances, where renowned organizations have witnessed such attacks, that has lasted for a longer time, many organizations are ramping up, leveraging the use of several algorithmic models to mitigate these attacks. Some of the infamous DDoS attacks are summarized in Table 1. In [20] An intrusion deduction system methodology was used to predict the malicious DDOS attacks in the SDN adaptation by using Machine Learning Algorithms namely, the Random Forest, Support Vector Machines, Decision tress and The Multiple Layer Perceptron. Scapy tool was used to launch the attack and Mininet tool with a POX controller tool performed the simulation. In [21] a classification algorithm and an entropy-based method was implemented, wherein, a two-class classification task distinguished normal flows from attacks.

Table 1. Some notable DDoS attacks in the past

Organization	Description of the attack
Amazon [22], [23], 2020, February	The attack was the largest attack, that had an intensity of 2.3Tbps. Relied on attacking CLDAP servers, to amplify the traffic. AWS shield, was successfully used for mitigation.
GitHub [24], 2018, February	This attack did not involve botnets, but a technique called meme caching was used, wherein a vulnerable server is a target and receives a

	spoofed request, that can intensify the traffic. The attack lasted for about 20 minutes, but was shortened owing to better mitigation plans and preparedness.
DYN [25], 2016, October	An organization, that provided network database services to tech giants, massive botnets were created to launch an attack through interconnected IOT devices.
BBC [26], 2015, December	A group called "New World Hacking" launched a 600 Gbps attack, and the tool used to launch it involved using cloud computing resources from two AWS servers.

Apart from DDoS attacks, ransomware attacks are a widespread threat that poses risks to the confidential data secured in a network infrastructure. In such an attack, the attacker deploys various methodologies to capture the confidential data of the user to confiscate hefty amounts from the user. Such an attack, is more severe in intensity due to the financial losses incurred to the industries [27]. There are two different forms of Ransomware attack which are Locker and crypto ransomware [28]. Such attacks, impact the network infrastructure negatively and in turn confiscates confidential data. [29] talks about the usage of intelligent algorithms in detecting ransomware attacks. Such a methodology classifies the algorithms as random forests, decision trees and deep learning algorithms. In [30], Random Forest classifier potentially identifies threats in windows, Virtual Box and other android based operating systems. In [31] decision tree algorithms were proposed to detect ransomware in windows. Such an algorithm works by mining the features of the infected network through its traffic conversations. In [32] decision tree was used to detect multiple variants of ransomware attacks. In [33] a hybrid model comprising of Classical Auto Encoder (CAE), a Variational Auto Encoder (VAE) and a deep neural network with batch normalization detected ransomware threats in the Internet of Things (IoT).

3 SDN Architecture

This section explains about a typical software-defined networking architecture with its various components and layers. In addition, an account on the HPE VAN SDN Controller with its essential features is also provided.

A typical SDN Architecture comprises of the following components. Figure 1 shows a simple schematic representation of the components and the layers of the SDN architecture.

a) **Applications:** These are programs that are used to directly convey the network requirements and its characteristics to the central repository (SDN controller) via an interface.

b) **Controller:** It holds the responsibility of providing a view of the entire network layers to the applications.
c) **Datapath:** Expresses complete control of incoming and forwarding data in the form of packets across the entire network.
d) **Data Plane:** An intermediary interface between the controller and the datapath, that apart from facilitating communication, is responsible for processing data transfers and notifying the network about incoming packets and transfers.
e) **North-bound interfaces:** Provides an entire overview of the network and analyses the communication and requirements of the network.
f) **South-bound interfaces:** Enables communication between controllers and switches and other network nodes, which is with the lower-level components of the SDN.

The centralized controller assumes the responsibility of the control plane and distributes the flow of data onto the entire network. The OpenFlow protocol, between the control and the data plane facilitates the communication between the data plane and the control plane [34]. SDN experiments are usually stimulated out on an open-source platform, The Mininet, an application that runs in a virtualized environment. This tool assists with creating new topologies and networks in a virtualized environment, and allows users to visualize the live functioning and experimentations on a network at no costs, as the tools are open-source [35]. Since it runs on a virtualized network, it may be slower than the actual ones. SDN networks, enables users to interact with the network by creating topologies, and customizes it in a manner to detect any interactions in the network [36]. The Software-Defined Networking (SDN) architecture is built in a way that makes it unable to independently manage the network by itself.
The architecture relies on a software called as a SDN controller, which manages the network with the aid of network administrators. In other words, a SDN controller is an application in the SDN environment that manages the flow control for better management of the network and improved application performance. The SDN controller is usually deployed on a server where it runs and manages several protocols to mediate the network traffic between switches and hosts. In addition, the controller software is responsible for communicating with applications with northbound APIs and the network infrastructure with southbound APIs. In essence, SDN controller acts like the “brain” of a software-defined network. To perform all the necessary tasks of the controller, a SDN Controller Platform is required, which is essentially a set of assembled software modules. This is illustrated with an example in Figure 2, wherein we have considered the control plane of the SDN architecture.

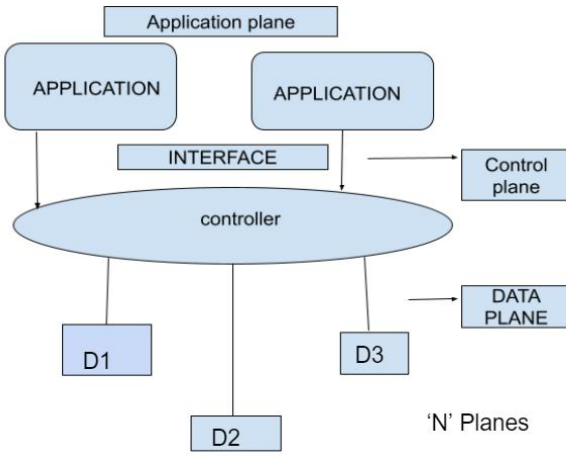


Fig. 1. Layers of SDN Architecture

There are numerous SDN controllers used for dictating the behaviour of the network. Some of the prominent ones include ONOS [37], OpenDayight [38], RYU [39] and OpenMUL [40]. However, we discuss the features of am OpenFlow-enabled network controller, namely the HPE VAN SDN Controller, which shall be used for the experiment in this paper. The HPE VAN SDN Controller acts as a single point of control in an OpenFlow-enabled network, simplifying maintenance, provisioning, and orchestration while allowing the delivery of a new generation of application-based network services. The control and data planes of the network are separated in the Hewlett Packard Enterprise Software Defined Networking (SDN) architecture, centralizing network intelligence, and abstracting the underlying network infrastructure from applications. The industry-standard OpenFlow protocol is used by controller software to provide physical and virtual switches under its management. The visibility of network ports, connections, and topologies allows for centralized policy administration and more effective path selection based on a dynamic, global picture of the network. For both mobile clients and servers, this greatly simplifies the orchestration of multi-tenant systems and the enforcement of network policies.

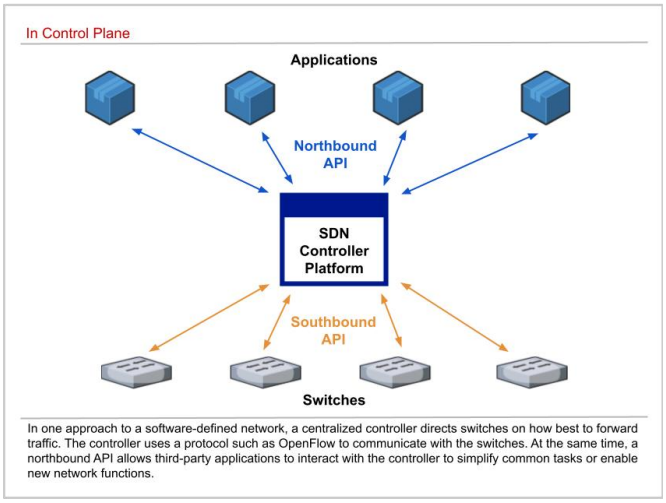


Fig. 2. SDN controller communicating with applications and switches using appropriate APIs

The HPE VAN SDN Controller is a premium software which is versatile in its usage. It is designed to operate in several computing environments including the service provider, public cloud, campus, private cloud, and data centers. Some of the highlight features of the controller are as follows:

a) Provides a platform for delivering a wide range of network improvements that is enterprise-class.
b) Provides a controller architecture that is extendable, scalable, and robust.
c) Compliance with OpenFlow 1.0 and 1.3 protocols.
d) Hewlett Packard Enterprise and H3C OpenFlow-enabled switches are supported.
e) Uses a local or distant Keystone server for secure authentication
f) Teams up controllers for a distributed platform scalability and high availability.
g) Common network services are provided by embedded applications.
h) Uses either specialized Java programs or general-purpose RESTful control interfaces, including functions to expand the controller REST API and UI, open APIs enable SDN application developers to create creative solutions that dynamically integrate business requirements to network infrastructure.
i) Integrates the HPE Intelligent Management Center (IMC) which manages and monitors the whole controller application life cycle, as well as providing better reporting and SDN network visibility [41].

The HPE VAN SDN Controller also provides a controller User Interface (UI) which acts the SDN Controller Platform for the users. The console UI can be used as follows:

a) View notifications and logs, as well as OpenFlow data including data flow details, topology of identified switches and end nodes, including shortest path, and OpenFlow classes that application programs have registered.
b) Acknowledge alerts, add, or enable applications, export log data, and enter licensing information, among other things.
c) Set key values for alert policies and other components of the SDN controller.
d) The SDN controller also has REST APIs that can be used to program or configure the controller, as well as construct applications that could be run on the controller.
e) There are some premium features (in the form of apps) offered by the controller that can be installed on the console from the HP SDN App Store.

4 Materials and Methods

To virtually demonstrate the effect of a DDoS attack in software-defined network, we shall need to create a virtual software-defined network first. In this section, we propose the materials and methods used to create a typical software-defined network. Section 4.1 describes the software technologies and tools that are used for creating such networks in contemporary times. The tools implemented concerning this research paper is also explained. Next, Section 4.2 and 4.3 highlights the advantages and limitations of the Mininet tool respectively.

4.1 Tools for creating a software-defined network

As far as this research paper is concerned, we discuss the features and functions of two software tools viz. The Mininet (a network emulator) and the HPE VAN SDN Controller (a SDN controller as discussed in Section 3), both of which shall be used in the topology creation process in Section 5. The reasons behind using these tools are:

- Mininet is an open-source tool which is easily accessible and can run on most of the Linux-based operating systems. For this research work the experiment was conducted in Linux environment.
- The commands in Mininet are easy to understand and implement.
- HPE VAN SDN controller is a user-friendly controller as compared to the rest.

Network emulators are software programs which runs on a standalone machine, and takes an abstract description of an actual network traffic. They are used by software developers for analyzing the response times, sensitivity to packet loss of clients and server applications and to emulate specific network access with different bit error rate, network dropouts, round-trip time, application dropouts and throughput. Emulators come in many different forms including integrated development environment appliances and browser-based network emulators. Some of the widely used network emulators include Mininet, UNetLab, Coolnix, Common Open Research Emulator (CORE), Marionnet and VNX. Since we shall be focusing on implementing a SDN network using the OpenFlow protocol, we choose Mininet as the network emulator to perform the experiments in this paper. Mininet is a network emulator which has been extensively used for creating realistic virtual networks. It is a useful tool for the creating virtual networks based on Software-Defined Networking using OpenFlow and P4. This enables us to experiment with and understand the working mechanism of the same. Besides, it can also be used for running switch, kernel, controller, links, and application code on a single laptop, computer, or any other machine (virtual machines, cloud). The Mininet hosts run on a standard Linux network software. Because, it is easier to learn the functioning of Mininet, with its interactive command line interface, ease of customization, sharing and deployment, Mininet has been used by many researchers, teachers, and development groups for building networks, prototyping, debugging, testing, and experimenting with custom experimental networks [42].

4.2 Significant characteristics of Mininet

a) Customizable, because it supports random custom network topologies, and consists of a basic set of parameterized topologies.
b) Extensible, because it a simple and extensible Python application programming interface (API) for creating and experimenting virtual networks. It also supports various other APIs.
c) Applicability, as the correct implementation source codes of prototype should be used in any arbitrary real network based on hardware requirements with negligible changes in the source code.
d) Interactivity, as the simulation of virtual networks should be comparable to real-time networks
e) Scalability, as the prototype networks must be capable to handle a vast network with numerous switches on a single machine.
f) Realistic, as the virtual prototype networks must resemble real-time networks with realistic behaviour, so that all applications and protocols can be used without any changes in the source code.

g) Shareable, as the created virtual networks should be easily shareable with other users or collaborators who can run the same and modify the experimental prototype as per their requirements.
h) Supporting concurrency, as it allows multiple developers to work independently on the same topology simultaneously.
i) Easily testable, as system-level regression tests, can be implemented with ease and get packaged. There is support of complex testing of the topology virtually, without any requirement of physical hardware [42], [43].

In addition to above characteristics, Mininet can be viewed as a package of useful features of various simulators, emulators, and hardware testbeds. In contrast to complete system virtualization procedure (emulator approach), Mininet boots faster, scales larger, provides more bandwidth, and can be installed easily. On the other hand, comparing to simulators, Mininet connects to realistic networks, runs practical and original code, and provides collaborative interface. Lastly, Mininet is better fit for use as compared to the hardware testbeds as it is inexpensive, always available, and can be easily reconfigured. It can also be easily restarted if any failure occurs.

Mininet works on the principle of process abstraction to virtualize the computer system resources. It runs several hosts and switches on a single operating system kernel using process-based virtualization. Network namespaces, a lightweight virtualization feature that enables individual processes with distinct network interfaces, routing tables, and ARP tables, have been supported by Linux since version 2.2.26. To enable complete OS-level virtualization, the full Linux container design adds "chroot()" jails, process and user namespaces, and CPU and memory limitations, however Mininet does not require these features. To connect switches and hosts, Mininet uses "virtual ethernet (veth) pairs." Furthermore, Mininet has a scope of supporting other operating systems with process-based virtualization as well, besides its current support for Linux kernel. Most of the code of Mininet is written in Python with a short implementation of C utilities [42].

4.3 Limitations of Mininet

a) It cannot run switches or other applications which are not Linux-based.
b) Though it runs on a single machine, it does impose resource limits of the system. For instance, it cannot surpass the CPU or bandwidth available on the given machine or server.
c) All the Mininet hosts share the PID space with the host file system, by default. This can cause some processes to be arbitrarily killed if care is not taken.

d) Since all the measurements re in real time, Mininet-based networks cannot be used to emulate faster-than-real-time results (e.g., networks with 100 Gbps or more bandwidth).
e) In order to create, custom switching or routing behaviour, Mininet lacks the ability to create a controlled for the user. One must arrange a controller if such networking behaviour is deemed necessary.

5 Topology creation

This section shows the usage of the Mininet tool in creating topologies using specific commands.
A network in Mininet can be created with a single CLI command. The network topologies are created using the "sudo mn" command. Diverse options can be appended with the "sudo mn" command such as "--topo=linear, 8". To integrate with the HPE VAN SDN Controller, the "--controller" option is used. In this case, the controller is running on a server with IP address 192.168.1.114;

```
~$ sudo mn --controller=remote,ip=192.168.1.114 --topo=linear,8
```

The above command creates a network of eight OpenFlow switches and eight hosts (PCs). The hosts need to send traffic so that they can be discovered by the SDN controller. This can be achieved using the "pingall" command in Mininet. The following command creates a network consisting of a single switch and eight hosts. All the eight hosts are connected to the same switch.

```
~$ sudo mn --controller=remote,ip=192.168.1.114 --topo=single,8
```

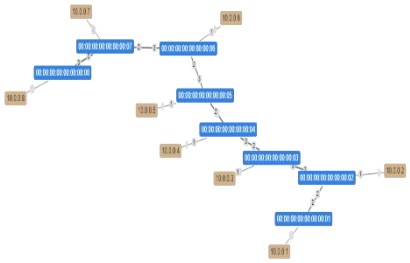
A single switch is used in a tree topology, with others connected to it based on a fanout number. A fanout value of 3 indicates that the core switch is connected to three switches, each of which may have three switches connected to it. This process is repeated based on the given depth. The number of hosts connected to each leaf or edge switch is also determined by the fanout value. The following command creates a network with tree topology with depth of 3 and fanout of 2. Each leaf switch has two hosts connected to it.

```
~$ sudo mn --controller=remote,ip=192.168.1.114 topo=tree,depth=3,fanout=2
```

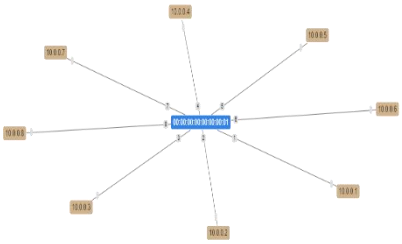
The following command creates a network with torus topology with dimensions 3x3.

```
~$ sudo mn --controller=remote,ip=192.168.1.114 --topo=torus,3,3
```

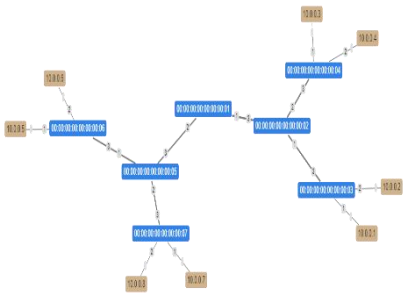
Figures 3(a - d) depict the linear, single, tree and torus topologies respectively, created by the using the aforementioned commands. The blue boxes represent the switches with their MAC addresses and the brown boxes represent the hosts with their IP addresses. The port numbers are also mentioned in the topology for each kind of network. The following figures are taken from the HPE VAN SDN Controller UI console.



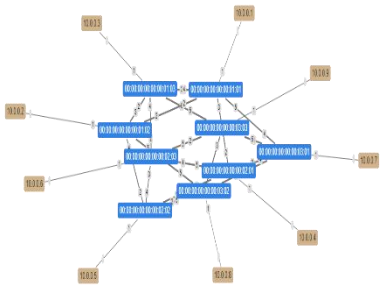
(a): Linear topology



(b): Single topology



(c): Tree topology



(d): Torus topology

Fig. 3. Typical linear, single, tree and torus topologies created on HPE VAN SDN Controller UI console

6 Methodology for Experimental Analysis

In this section, the methods for performing the experimental analysis for three different types of network topology are explained.

Two experiments were performed to analyze and compare the traffic flow in three different types of topologies viz. linear, single, and tree topology. The two experiments described in the sub-sections 6.1 and 6.2 was performed on a HP

Laptop 14s-cr1005TU PC with the following specifications used; Intel(R) Core (TM) i5-8265U CPU @ 1.60GHz 1.80 GHz, 8.00 GB of RAM running the Windows 10 (Version 21H1) 64-bit operating system and Oracle VM VirtualBox version 6.1.28.
The guest operating systems used on the VM: Linux operating system Ubuntu 14.04.4 LTS (64-bit) for the "mininet-vm tty1" with 1 GB of base memory. The HPE VAN SDN Controller was installed on Linux operating system Ubuntu 12.04.5 LTS (64-bit) with 1 GB base memory. Both the guest operating systems were managed by the VirtualBox Manager. Further, the HPE VAN SDN Controller UI console was accessed via https://192.168.1.114:8443/sdn/ui on Google Chrome Version 96.0.4664.45 (Official Build) (64-bit) web browser.

6.1 Experiment - Testbed - 1

In the first experiment, the total number of hosts in each type of network topology is set to 8. The network topologies are created using the following CLI commands:

```
~$ sudo mn –controller=remote,ip=192.168.1.114 --topo=linear,8

~$ sudo mn –controller=remote,ip=192.168.1.114 --topo=single,8

~$ sudo mn –controller=remote,ip=192.168.1.114 -topo=tree,depth=3,fanout=2
```

For each topology, 10 packets are sent between the source host (h1) and the destination host (h2) using the "h1 ping –c10 h8" command. As a result, ten ICMP packets with sequence number starting from 1 with 64 bytes of data is pinged from h1 to h8. The ping statistics are obtained at the destination host, which has been tabulated in Section 7.

6.2 Experiment - Testbed - 2

In the second experiment, for each type of topology the relationship between the number of nodes and average round-trip time (RTT) in milliseconds (ms) is observed. The number of hosts are increased exponentially; 2n where n = 1 to 8. For each case of simulation with a given number of nodes (hosts and switches), 5 packets were pinged from the first host to the last host in the network. The results are tabulated in Section 7.

7 Results and Discussion

For the first experiment, the ping statistics in terms of minimum, average, maximum, and minimum deviation round-trip time (RTT) in milliseconds (ms), for each of the three topologies is provided in the Table 2. The ping statistics of the first experiment is also graphically presented in Figure 4.

Table 2. Ping statistics for the first experiment

Topology	Minimum RTT (ms)	Average RTT (ms)	Maximum RTT (ms)	Min. Deviation of RTT (ms)
Linear	0.085	0.432	1.679	0.596
Single	0.054	0.230	1.568	0.446
Tree	0.046	0.299	1.368	0.429

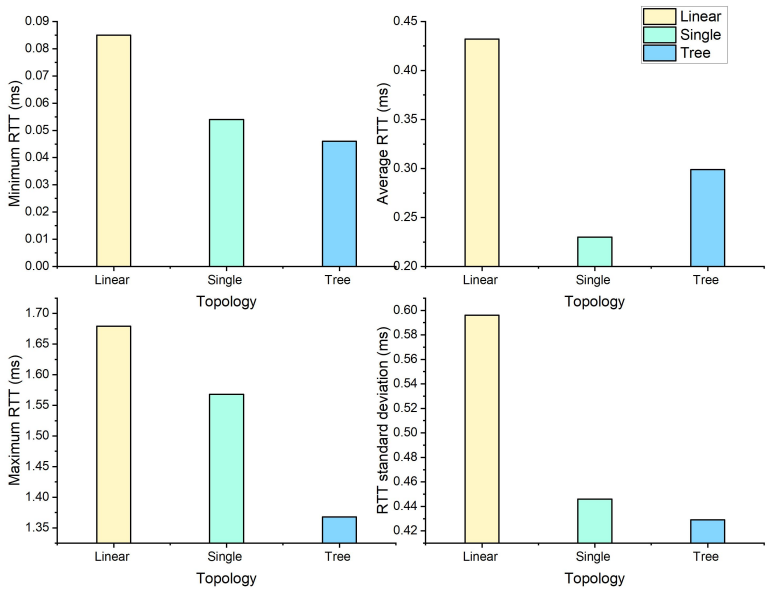


Fig. 4. Comparison of ping statistics between the linear, single, and tree topologies.

In the first quadrant of Figure 4, we observe that minimum RTT is the least for tree topology. In the second quadrant, the single topology has the least average RTT. Further in the third and fourth quadrants, the tree topology has the least maximum RTT and minimum deviation in RTT. The results obtained from the second experiment are shown in Table 3 for the tree topology, in Table 4 for the single topology and in Table 5 for the linear topology. The results for these three tables are visualized in Figures 5(a - c).

Table 3. Second experiment scalability results for the tree topology

Topology	Number of Nodes	Number of Hosts	Number of Switches	Average RTT (ms)
Tree	3	2	1	0.137
Tree	7	4	3	0.196
Tree	15	8	7	0.201
Tree	31	16	15	0.256
Tree	63	32	31	0.317
Tree	127	64	63	0.381
Tree	255	128	127	1.423
Tree	511	256	255	2.479

Table 4. Second experiment scalability results for the single topology

Topology	Number of Nodes	Number of Hosts	Number of Switches	Average RTT (ms)
Single	3	2	1	0.157
Single	5	4	1	0.158
Single	9	8	1	0.135
Single	17	16	1	0.122
Single	33	32	1	0.133
Single	65	64	1	0.111
Single	129	128	1	0.123
Single	257	256	1	0.120

Table 5. Second experiment scalability results for the linear topology

Topology	Number of Nodes	Number of Hosts	Number of Switches	Average RTT (ms)
Linear	4	2	2	0.130
Linear	8	4	4	0.182
Linear	16	8	8	0.257
Linear	32	16	16	0.814
Linear	64	32	32	1.067
Linear	128	64	64	2.096
Linear	256	128	128	3.147
Linear	512	256	256	5.141

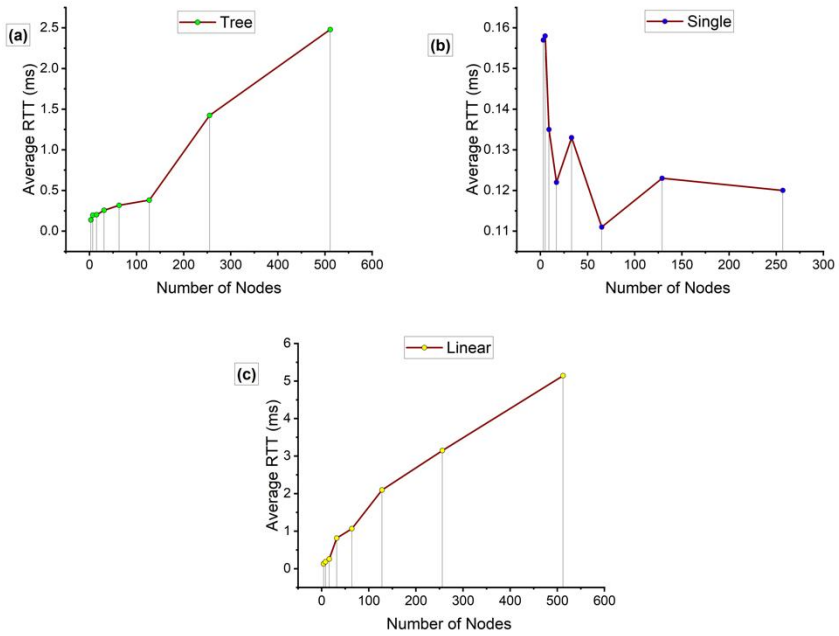


Fig. 5. Graphical analysis between number of nodes and average round-trip time for the three topologies.

From the Figures 5(a) and 5(c), we observe that as the number of nodes increase the average RTT also increases sharply. But in Figure 5(b), there is overall decline in average RTT as the number of nodes increase.

8 Conclusion and Future Work

Through this paper, we have analyzed the features of the tools namely, Mininet and HPE VAN SDN controller. The architecture of an SDN controller was analyzed in detail, and different topologies such as linear, tree topology, were simulated using the commands in the console. Several parameters were derived from the simulation, such as average, minimum and maximum round trip Time in milliseconds. The ping statistics were visualized by plotting a graph, that compares the topology and the round-trip time and the second experimental analysis visualized a graphical relationship between the number of nodes and the average round trip time.

The future work for this research work can be aimed towards the actual simulation of distributed denial of service attacks in the virtual networks created by Mininet. Besides, we can also analyze the behaviour of DDoS attacks in varying topologies of the virtual software-defined network in its control plane. Several

other tools could also be incorporated besides the one used in this research work to choose the attack node and simulate the attack successfully. Ransomware attacks could also be studied using similar topologies. Lastly, the sophisticated torus topology still needs to be researched thoroughly to study its behaviour in terms of round-trip time and the behaviour of DDoS attack in similar kinds of complex topology.

References

1. Ottis, R.: Analysis of the 2007 cyber attacks against Estonia from the information warfare perspective. In: Proceedings of the 7th European Conference on Information Warfare, p. 163 (2008)
2. Hoque, N.; Bhattacharyya, D.; Kalita, J.: Botnet in DDoS attacks: trends and challenges. IEEE Commun. Surv. Tutor. 99, 1–1 (2015)
3. Bawany, N.Z., Shamsi, J.A. and Salah, K., 2017. DDoS attack detection and mitigation using SDN: methods, practices, and solutions. Arabian Journal for Science and Engineering, 42(2), pp.425-441.
4. Jain, S.; et al.: B4: experience with a globally-deployed software defined WA. ACM SIGCOMM Comput. Commun. Rev. 43(4), 3–14 (2013).
5. Douligeris, C., & Mitrokotsa, A. (2004). DDoS attacks and defense mechanisms: classification and state-of-the-art. Computer Networks, 44(5), 643–666.
6. P.J. Criscuolo, Distributed Denial of Service Trin00, Tribe Flood Network, Tribe Flood Network 2000, and Stacheldraht CIAC-2319, Department of Energy Computer Incident Advisory (CIAC), UCRL-ID-136939, Rev. 1, Lawrence Livermore National Laboratory, February 14, 2000, Available from <http://ftp.se.kde.org/pub/security/csir/ciac/ciacdocs/ciac2319.txt>.
7. D. Dittrich, The DoS Projects "trinoo" Distributed Denial of Service attack tool, University of Washington, October 21, 1999, Available from http://staff.washington.edu/dittrich/misc/trinoo.analysis.txt
8. D. Dittrich, The Tribe Flood Network Distributed Denial of Service attack tool, University of Washington, October 21, 1999
9. J. Barlow, W. Thrower, TFN2K—an analysis, 2000 Available from .<http://security.royans.net/info/posts/bugtraq_DDoS2.shtml>.
10. CERT Coordination Center, Center Advisory CA-1999-17 Denial of Service tools ,Available from . m <http://www.cert.org/advisories/CA-1999-17.html>.
11. C. Adams, J. Gilchrist, The CAST-256 encryption algorithm, RFC 2612, June 1999, Available from <http://www.cis.ohio-state.edu/htbin/rfc/rfc2612.html>
12. D. Dittrich, G. Weaver, S. Dietrich, N. Long, The mstream Distributed Denial of Service attack tool, May 2000, Available from <http://staff.washington.edu/dittrich/misc.mstream.analysis.txt>
13. D. Moore, G. Voelker, S. Savage, Inferring Internet Denial of Service activity, in: Proceedings of the USENIX Security Symposium, Washington, DC, USA, 2001, pp. 9–22.

14. Q. Yan, F. R. Yu, Q. Gong and J. Li, "Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges," in IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 602-622, Firstquarter 2016, doi: 10.1109/COMST.2015.2487361.
15. Salman, O., Elhajj, I. H., Kayssi, A., & Chehab, A. (2016). SDN controllers: A comparative study. 2016 18th Mediterranean Electrotechnical Conference (MELECON). doi:10.1109/melcon.2016.7495430
16. A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado and R. Sherwood, '"On controller performance in software-defined networks," USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE), vol. 54. 2012.
17. R. Khondoker, A. Zaalouk, R. Marx and K. Bayarou, '"Featurebased comparison and selection of Software Defined Networking (SDN) controllers," Computer Applications and Information Systems (WCCAIS), 2014 World Congress on, pp. 1-7.
18. A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov and R. Smeliansky, '"Advanced study of SDN/OpenFlow controllers," Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, ACM, 2013, pp. 1.
19. Kalkan, K., Gur, G. and Alagoz, F., 2017. Defense mechanisms against DDoS attacks in SDN environment. IEEE Communications Magazine, 55(9), pp.175-179.
20. Santos, R., Souza, D., Santo, W., Ribeiro, A., & Moreno, E. (2019). *Machine learning algorithms to detect DDoS attacks in SDN. Concurrency and Computation: Practice and Experience, e5402.* doi:10.1002/cpe.5402
21. Banitalebi Dehkordi, A., Soltanaghaei, M., & Boroujeni, F. Z. (2020). *The DDoS attacks detection through machine learning and statistical methods in SDN. The Journal of Supercomputing.* doi:10.1007/s11227-020-03323-w
22. "Amazon says it mitigated the largest DDoS attack ever recorded - The Verge." https://www.theverge.com/2020/6/18/21295337/amazon-aws-biggest-DDoS-attack-ever-2-3-tbps-shield-github-netscout-arbor (accessed Feb. 08, 2022).
23. "Amazon 'thwarts largest ever DDoS cyber-attack' - BBC News." https://www.bbc.com/news/technology-53093611 (accessed Feb. 08, 2022).
24. "DDOS attacks and the GitHub case » IRIS-BH." https://irisbh.com.br/en/DDoS-attacks-and-the-github-case/ (accessed Feb. 08, 2022).
25. "Cyber Case Study: The Mirai DDoS Attack on Dyn - CoverLink Insurance | Ohio Independent Insurance Agency." https://coverlink.com/case-study/mirai-DDoS-attack-on-dyn/ (accessed Feb. 08, 2022).
26. "DDoS attack on BBC may have been biggest in history | CSO Online." https://www.csoonline.com/article/3020292/DDoS-attack-on-bbc-may-have-been-biggest-in-history.html (accessed Feb. 08, 2022).
27. Bello, I., Chiroma, H., Abdullahi, U. A., Gital, A. Y., Jauro, F., Khan, A., … Abdulhamid, S. M. (2020). Detecting ransomware attacks using intelligent algorithms: recent development and next direction from deep learning and big data perspectives. Journal of Ambient Intelligence and Humanized Computing. doi:10.1007/s12652-020-02630-7

28. Reshmi, T. R. (2021). Information security breaches due to ransomware attacks - a systematic literature review. International Journal of Information Management Data Insights, 1(2), 100013. doi:10.1016/j.jjimei.2021.100013
29. Digital Guardian (2019) A history of ransomware attacks: the biggest and worst ransomware attacks of all time. https://digitalguardian .com/blog/history-ransomware-attacks-biggest-and-worst-ranso mware-attacks-all-time. Accessed 17 Dec 2019
30. Agrawal R, Stokes JW, Selvaraj K, Marinescu M (2019) Attention in recurrent neural networks for ransomware detection. In: Paper pre sented at the ICASSP 2019–2019 IEEE international conference on acoustics, speech and signal processing (ICASSP).
31. Alhawi OM, Baldwin J, Dehghantanha A (2018) Leveraging machine learning techniques for windows ransomware network trafc detection. Cyber Threat Intell 70:93–106
32. Alrawashdeh K, Purdy C (2018) Ransomware detection using limited precision deep learning structure in fpga. In: Paper presented at the NAECON 2018-IEEE national aerospace and electronics conference.
33. Cusack G, Michel O, Keller E (2018) Machine learning-based detection of ransomware using SDN, pp 1–6. https://doi.org/10.1145/31804 65.3180467.
34. "Open Networking Foundation," https://www.opennetworking.org/sdndefinition (accessed Nov. 12, 2021).
35. Shie-Yuan Wang, "Comparison of SDN OpenFlow Network Simulator and Emulators: EstiNet vs. Mininet."
36. B. Lantz, B. Heller, and N. Mckeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks".
37. "Open Network Operating System (ONOS) SDN Controller for SDN/NFV Solutions." https://opennetworking.org/onos/ (accessed Nov. 12, 2021).
38. "Platform Overview - OpenDaylight." https://www.opendaylight.org/about/platform-overview (accessed Nov. 12, 2021).
39. "Ryu SDN Framework." https://ryu-sdn.org/ (accessed Nov. 12, 2021).
40. D. Saikia, N. Malik Jaffe, and T. White Paper, "Whitepaper Openmul An Introduction to OpenMUL SDN Suite," 2014, Accessed: Nov. 12, 2021. [Online]. Available: www.openmul.org
41. "Introduction to the HPE VAN SDN Controller." https://techhub.hpe.com/eginfolib/networking/docs/sdn/sdnc2_7/5200-0910prog/content/c_sdnc-pg-intro.html (accessed Nov. 12, 2021).
42. "Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet." http://mininet.org/ (accessed Nov. 11, 2021).
43. Faris Keti, Shavan Askar, "Emulation of Software Defined Networks Using Mininet in Different Simulation Environments". In proceedings of the 6th International Conference on Intelligent Systems, Modelling and Simulation, 2015.