CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Introduction

This chapter discusses the methodology of evaluating the proposed scheme, elapsed-time based scheme. In order to meet the research objectives, the methodology acts as guidelines to carry the research out in four phases. Each phase has its own goal. The goal of phase one is reviewing related literature regarding DDoS detection and mitigation in SDN, studying of DDoS detection techniques namely machine learning and entropy techniques, identifying the issues related to the performance of the machine learning and entropy techniques particularly, overhead and accuracy and analyzing the parameters used by previous works that used these techniques to identify parameters that contribute to decrease the overhead and increase the accuracy. In phase two, the drawbacks of the proposed techniques are addressed and a design of a new scheme for detecting and mitigating DDoS attacks in SDN is proposed to fill up the gap in the literature. In phase three, the implementation of scheme design is illustrated in details. In phase four, testing of the implemented scheme's performance in terms of overhead, accuracy and false alarms is finally evaluated. The phases of the research methodology are illustrated in Figure 3.1.

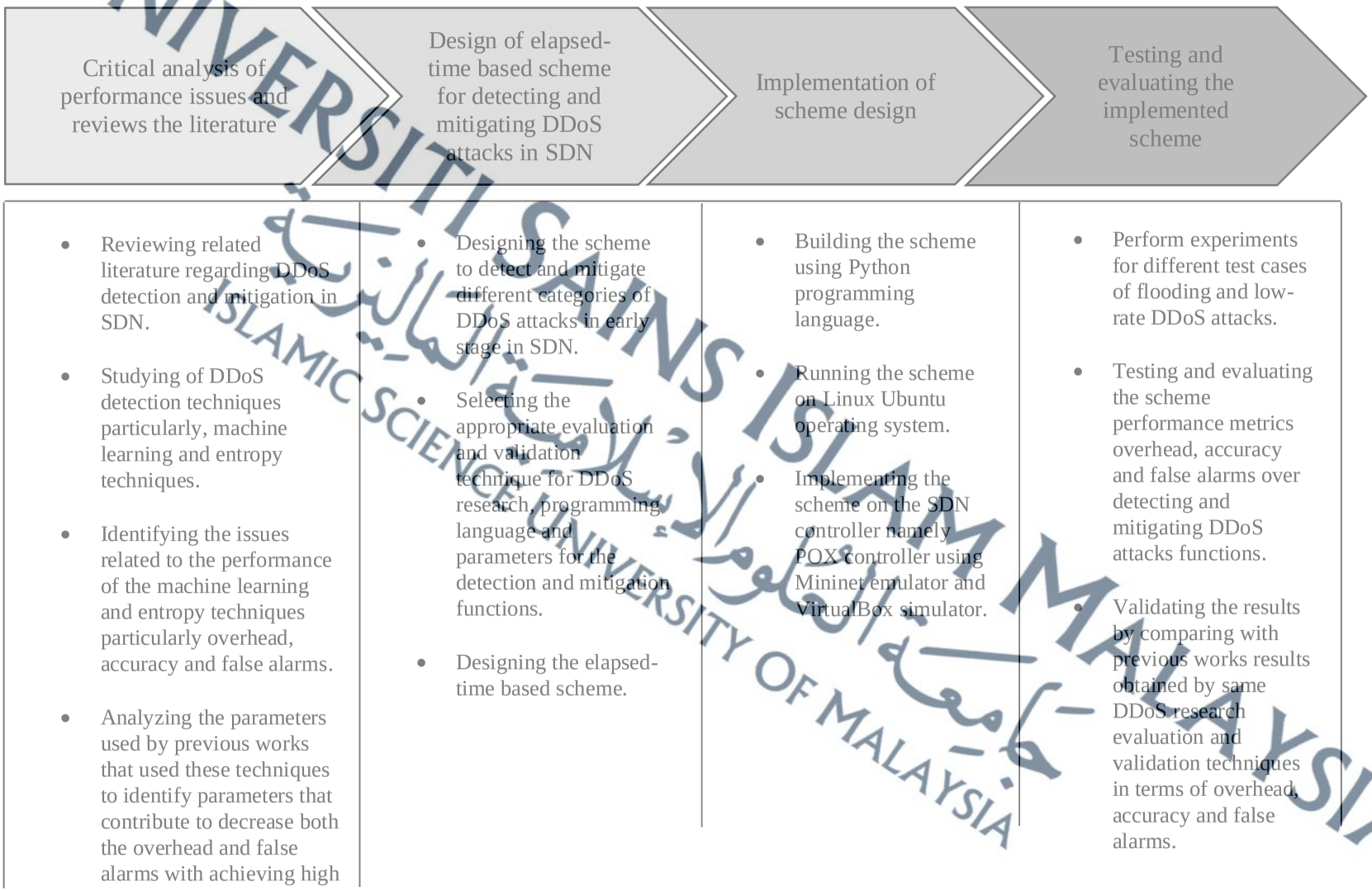

Figure 3.1: Research methodology

3.2 Designing the elapsed-time based scheme

The proposed scheme is meant to provide an accurate and less complex solution that has low false alarms for detecting and mitigating flooding DDoS, low rate DDoS and change volume DDoS. To achieve the purposes of designing the scheme, we need to select the appropriate validation technique for a DDoS research, simulator and emulator software, appropriate programming language and appropriate parameters to be used in the detection and mitigation functions.

3.2.1 Selecting a validation technique for a DDoS research

In networks, performance evaluation approaches generally classified into three main techniques: mathematical models, measurement, and simulation (Jain, 2015). In networks security particularly DDoS research domain, techniques used for evaluation and validation are four techniques.

According to (Behal & Kumar, 2016), "whenever a researcher proposes any novel detection or defense method in the field of network security, the proposed method has to be implemented in the form of a network based experiment for its evaluation and then it needs to be validated through available set of validation techniques. There are basically four approaches used for validation in network based experiments: mathematical models, simulation, emulation and real systems" (p. 9). Figure 3.2 represents these techniques.

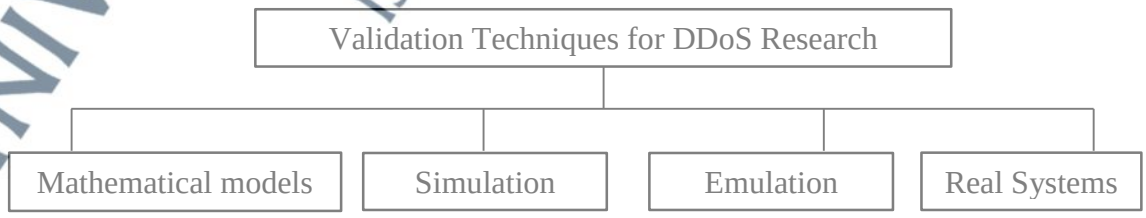

Figure 3.2: Validation techniques (Adopted from Behal and Kumar, 2016)

3.2.1.1 Mathematical Models

Due to that mathematical models are theoretical in nature, the given system, applications, platforms and conditions are modelled symbolically and then validated mathematically. For that, mathematical models can't be used for such network based experiments as pointed out by Behal and Kumar (2016). For the purpose of this research, this technique is rendering not suitable.

3.2.1.2 Simulation

The method of using computer software to model a real world process operation, event, or system is known as simulation (law, 2007). Simulation is a significant methodology approach that has been increasingly used in theory development (Davis et al., 2007). According to (Ivanov, 2017), simulations have been used in numerous effective research studies as their primary research methodology. In addition, simulation provides a very better insight into complicated environments without the need to construct these time consuming, expensive and sometimes risky real environments. According to (Gao & Chen, 2016), simulation can be considered as a methodology to further enhance the optimization results or it can be even used as a simulation-based optimization technique. Moreover, simulation provides an analytically accurate means to understand the behaviour of any element in the simulated environment (Fu et al., 2005).

However, the simulation technique can't execute real applications as it can approximate certain hardware and software functions only.

3.2.1.3 Emulation

The integration of simulation and real systems is the emulation. In emulation, combining real elements of an operating system and real applications with unreal and simulated elements like soft network links, virtual intermediate nodes, and unrealistic background traffic is made. While the simulation runs in virtual simulated time, emulation runs in real time.

3.2.1.4 Real Systems

Due to providing realistic network conditions, real operating systems, applications and platforms; real systems technique is proven to be best for network based experimentations.

However, there are some limitations of using real systems for experimentations. Firstly, changing the network topology is not possible for a new experiment. Secondly, it is very unsafe to do live experiments with Internet intrusions because they can easily escape from the experimentation setup and can damage the live network components. Finally, the flooding based DDoS attacks can cause degradation of network links (Schmidt et al., 2010). Furthermore, this technique is very costly when it comes to providing the equipment and tools (Ince & Bragg, 2007). Real systems are ideal for validating network based research but they are limited in their functionality because of their complex nature. However, in this research, it is beyond our capacity to use this technique even though it can reflect the behaviour of the proposed scheme.

Due to that there is no appropriate dataset available for validating related DDoS research among the various publically available datasets because they are

obsolete, lack the required characteristics of network traffic or not made available to the research community for security reasons (Behal & Kumar, 2016), the appropriate evaluation and validation techniques for this research are simulation and emulation.

3.2.2 Selecting a Network Simulator and Emulator

Selecting a network simulator is an important decision to make, because choosing the appropriate choice will save time and efforts, and will definitely lead to efficient results. Selecting a network simulator can be done by taking into account the modelling capabilities and hardware and software consideration.

Modelling capabilities include how well and accurately the simulator can represent the topology that has been created. Main considerations here are the educational and experience level required to use a particular simulator in terms of friendless issues and the ease of learning.

On the other hand, the hardware and software considerations include how flexible the simulation software to model the simple as well as complex systems. This category includes programming features supported by the programming language such as built-in functions and object-oriented programming and others. Furthermore, the ability to run the simulator on newer or older hardware without the need for using specific hardware plays a major role in selecting the network simulation software. In addition, the cost of purchase, installation and maintenance is another major consideration for selecting the proper simulator.

In order to select a proper simulator, it is very important to have a very good knowledge of the simulation software and its tools. For the purpose of this research, we use VirtualBox simulator. This simulation software was selected according to its

popularity and its capabilities. The following section describes briefly the selected simulator.

3.2.2.1 VirtualBox Software

VirtualBox belongs to Oracle Company and it is named as Oracle VM VirtualBox. VirtualBox is a virtualization application which used as a cross-platform to allow you to run multiple operating systems inside multiple virtual machines at the same time. VirtualBox is a free and open-source solution that works with all platforms including Windows, Mac, Linux, and Solaris. For example, you can run Linux OS on your Windows computer or Linux and Windows on Mac and so on, all alongside with your current running applications. VirtualBox can be running on different host operating systems whether 32-bit or 64-bit operating systems and also on different versions from the same operating system. Furthermore, VirtualBox does not require hardware virtualization; therefore you can run it on an older hardware where the features built into newer hardware are not present.

As previously mentioned, the core, open-source VirtualBox package is free under general public use (GPU) license, and its proprietary extension package is free indefinitely under a personal use and evaluation license (PUEL). VirtualBox offers many of features and these features are listed below:

- Cross-platform compatibility (installs on Mac, Linux, Windows, Solaris computers)
- Command line interaction
- Shared folders and clipboard
- Special drivers and utilities to facilitate switching between systems

- Snapshots
- Seamless mode (lets you run virtual applications next to normal ones)
- Limited support for 3D graphics (up to OpenGL 3.0)
- Can exchange disk images with VMware
- VM video capture
- VM disk image encryption (with extension pack)
- Virtual USB 2.0/3.0 support (with extension pack)

Unlike traditional network, SDN structure needs emulator software to emulate the OpenFlow components (e.g. controllers, switches and hosts). Because emulating large networks with large numbers of hosts, switches and SDN controllers, using the Internet may not be a good idea, improper configurations can cause unwanted problems. To solve this problem, Mininet software has been created. The following section describes briefly the SDN emulator.

3.2.2.2 Mininet Emulator

Mininet was created at Stanford University by a group of professors as a tool for teaching network technologies and research purposes. Mininet is a network emulation orchestration system which creates and runs a network of virtual end-hosts, switches, controllers, routers, and links. Mininet components behave just like real machines on a complete network where the components can communicate with each other smoothly. The programs on Mininet can send and receive packets through interfaces look like real Ethernet interfaces with a speed and delay of a link. These packets can be processed by what seems like real routers and switches. In short, Mininet is the easy way to create virtual software defined networking.

Mininet supports many tasks that can benefit from having a complete experimental network on a laptop. Mininet supports development where it provides a network testbed for the OpenFlow applications development which is simple and inexpensive. Mininet supports research and learning purposes where the researchers can apply their research experiments on a ready, simple and complete network environment without the need of expensive labs. Also, Mininet supports topologies testing where it enables the complex topology testing without the need for a wired-up network. Mininet includes the Command Line Interface (CLI) for debugging and Python API for network creation and experimentation. Mininet allows Python-based user scripts to interface with it because it is written in Python. On Mininet you can run the real programs that run on Linux such as the TCP window monitoring tools and the network packet analyzer Wireshark. Many virtual machines (VMs) with different operating systems such as Linux and Windows can be created on Mininet VM which uses Linux Ubuntu OS. (Version 2.2.1; Mininet). For the purpose of this research, we have selected VirtualBox simulator and Mininet emulator to conduct the topology presented in SDN environment.

3.2.3 Selecting SDN Controller

In a typical SDN, the network intelligence is logically centralized in controllers (software-based), which enables the control logic to be designed and operated on a global network view, as a centralized application, rather than a distributed system. A program running on a logically-centralized controller manages the network directly by configuring the packet-handling mechanisms in the underlying network devices. The packet processing rules are installed on network devices to realize various tasks

of managing a network, ranging from routing and traffic monitoring to access control and server load balancing.

The separation of control and data plane, a decoupled system, has been compared to an operating system, in which the controller provides a programmatic interface to the network that can be used to implement management tasks and offer new functionalities. There are various open-source and commercial controllers in the market. However, our focus in this research has been given to open-source controllers. Some of the open-source controllers are Beacon, Floodlight, POX, and Ryu. These controllers have been described in section 2.5 in literature review chapter. Also, a review of their characteristics has been given in Table two in the same section. In the following section we will describe the SDN POX controller.

3.2.3.1 POX Controller

Controlling the network, managing its resources, providing its configurations and sending and receiving the forwarding instructions to and from the network forwarding equipment is the function of software-defined networking controller. The controller is located in the middle layer in SDN which is control layer. SDN has many types of controllers made with different types of programming languages. POX controller is a Python-based controller and it comes already installed on Mininet.

POX is an open source controller provides communication with SDN switches using the OpenFlow protocol. The general purpose of the controller is to allow developers to write their own applications that use the controller either as an intermediary or as an abstraction layer (Fernandez, 2013). The abstraction layer means a layer between network applications and network equipment. Furthermore,

POX controller comes with stock components bundled with it to allow developers to create a more complex SDN controller by creating new POX components or writing network applications that address POX Python Application Programming Interface (API). We have chosen POX controller to be used in our experiments because Mininet is written in Python. The POX controller version used in this experiment is (0.5.0 eel) which is the current version in use.

3.2.4 Selecting a Programming Language

Typically, controllers in SDN are classified into two classifications based on the programming language that they constructed by. Precisely, SDN controllers can be classified to Python-based controllers and Java-based controllers. Because the POX controller is a Python-based controller and it has been nominated to be the controller used for all experiments in this research, therefore, the programming language that will be used to construct the scheme is Python.

3.2.4.1 Python Programming Language

Python is a powerful high-level programming language that supports features like functional programming and object-oriented programming. From our perspective, Python is one of the easiest computer languages due to ease of using its syntax which makes it the perfect language to understand, learn, and use. Some of its advantages are listed below:

- Easy Syntax
- Readability
- High-Level Language
- Object oriented programming
- It is free

- Cross-platform
- Widely Supported
- It is Safe

Python's syntax is easy to learn, so both non-programmers and programmers can start programming right away. Furthermore, Python's syntax is very clear, so it is easy to understand program code. Python is often referred to as "executable pseudo-code" because its syntax mostly follows the conventions used by programmers to outline their ideas without the formal verbosity of code in most programming languages. In other words, syntax of Python is almost identical to the simplified "pseudo-code" used by many programmers to prototype and describes their solution to other programmers. Thus, Python can be used to prototype and test code which is later to be implemented in other programming languages.

Python is both free and open-source. Therefore, it can run on all major operating systems like Microsoft Windows, Linux, and Mac OS X. Python has an active support community with many web sites, mailing lists, and internet groups that attract a large number of knowledgeable and helpful contributes. Also, Python does not have pointers like other C-based languages, making it much more reliable. Along with that, errors never pass silently unless they are explicitly silenced. This allows you to see and read why the program crashed and where to correct your error.

3.2.5 Selecting Parameters for Detecting and Mitigating DDoS in SDN

Selecting the appropriate parameters is the important step to accomplish the task of detecting DDoS attack. Therefore, the full understanding of the attack category plays a major role in selecting the effective parameters.

3.2.5.1 Parameters Used for DDoS Detection

Based on the critical analysis conducted in the literature review chapter, we found that solutions proposed for the purpose of detecting DDoS attacks have used different parameters but the arrival time of packets was not among them. The analysis conducted also shows that the performance of the proposed solutions had been affected, particularly the overhead and false alarms. Also, the analysis conducted in chapter two shows that proposed DDoS detection solutions have not proposed to detect either the attacks that change from the high volume attack to the low volume or the low rate attacks. Therefore, we are of the opinion that selecting the appropriate parameters particularly the elapsed time between successive attack packets and the number of flows will reflect positively on detecting different categories of DDoS attacks and enhancing the performance at the same time.

By looking at the nature of the flooding DDoS attack, we can see that this attack is generated in a massive amount of traffic from different sources locations known as compromised computers towards a certain destination called the victim. The nature of the attack requires that this large number of packets be continuously generated and fast sent. The idea behind the fast and continuous sending is to overwhelm the target machine by these packets to make the target idle or completely unavailable to legitimate users. Based on this nature, the time elapsed between the successive attacks packets should be a few seconds. The elapsed time between the successive attack packets is calculated by subtracting the arrival time of each two successive packets. On the contrary to other research studies, we have used this parameter as a key parameter in detecting flooding DDoS attacks in SDN. We

strongly believe that using this parameter is the successful step to performance enhancement.

To accomplish the task of detecting flooding DDoS attacks, other parameters should be selected. Based on the description given on the nature of the attack in the previous paragraph, packets-count (the number of packets) is another appropriate parameter that has to be selected.

OpenFlow provides several functionalities in SDN; updating packets counter every time a packet arrives at the switch is one of them. This functionality provides the total number of packets that currently existed in the network. The packets counter is one of the flow entries in the flow table in the OpenFlow switch. Flow entries were illustrated in Figure 2.3 section 2.4.1 in chapter two.

OpenFlow also provides the functionality of extracting the header of each new packet and forwarding it to the controller to instruct the switch about what actions should be taken. The header of each packet contains the source and destination IP addresses, source and destination ports and protocol number. Due to targeting a particular destination by the DDoS attacker, the destination IP address is a primary parameter to select.

From my point of view, another important parameter to select is the total number of flows (flow-count). The total number of flows is important because of the role it plays to detect the attack packets when the attacker moves from the high volume attack to the low volume at the time of the attack. In this case, the number of attack packets is sharply decreased to escape the threshold and to appear as normal packets. This small number of packets generates a large number of flows to contain

these packets in the switch. So, selecting flow-count as a parameter will be an appropriate choice to notify the controller with the low volume of attack packets. Furthermore, flow-count is an important parameter because it is the proper choice to notify the controller with low rate attacks. In this case, the attacker generates the attack packets at a low rate and sends them in a short or a long duration to increase the number of flows in the switch. This attack sends the packets separately to overflow the switch with a large number of flows. Therefore, this group of chosen parameters will allow detecting different types of DDoS attacks in the early stages with enhancing performance.

3.2.5.2 Parameters Used for DDoS Mitigation

Attack detection is the essential and important step to identify the malicious packets but it is not enough to avoid the attack damage. Due to this, the attack detection process must be followed by an attack mitigation step. Attack mitigation is required to eliminate the malicious packets to protect the network resources from being unavailable or in the worst scenarios crash the network entirely.

The flow table in the OpenFlow switch consists of flow entries such as match fields, priority, counters, instructions, timeouts, cookie and flags. Precisely, each flow entry has an idle timeout and a hard timeout associated with it, both of which are configured through the OpenFlow controller. These two flow entries are specifically designed to have the maximum amount of time or idle time before flow is expired by the switch.

The idle timeout is the number of seconds after which a flow entry is removed from the flow table and the hardware provided because no packets match it. The hard

timeout is the number of seconds after which the flow entry is removed from the flow table and the hardware whether or not packets match it.

If a flow entry has both an idle timer and a hard timer associated with it, the first timer to expire causes the flow entry to be removed. If the idle timer expires first, the flow entry is removed at that point only if there are no matching packets. Otherwise, the flow entry is removed when the hard timer expires.

When the controller sends a flow entry modification message (OFPT_FLOW_MOD) to the switch, it specifies the idle timeout and hard timeout for that flow entry.

On the contrary to other mitigation techniques, we use the flow entry modification message to specify the idle timeout and hard timeout for the flow entry to remove this particular flow entry from the switch directly once the attack packets detected and dropped by the controller. Thus, the switch flow tables will be protected from being overflow by the attack requests and keep the storage capacity of the switch available to the new coming requests. Moreover, protecting the switch will lead to protect the controller from forwarding the unprocessed requests from the entire network when the switch is overflow.

3.2.6 Simulation Topology

As presented in section 3.3.1, the simulation and emulation techniques were used for evaluation and validation. The network simulator VirtualBox and the emulator Mininet were selected to perform all experiments in this research. The experiments were carried out by defining the simulation topology, running the simulation scenarios, and collecting, processing, and analyzing the results.

For the simulation topology, a network of one OpenFlow switch, one controller and six hosts was created using Mininet. The topology appeared in Figure 3.3. To evaluate the proposed scheme, we used three different experimental scenarios which contained different test cases and different DDoS attack types. Also, the proposed scheme has been evaluated in the short and long terms.

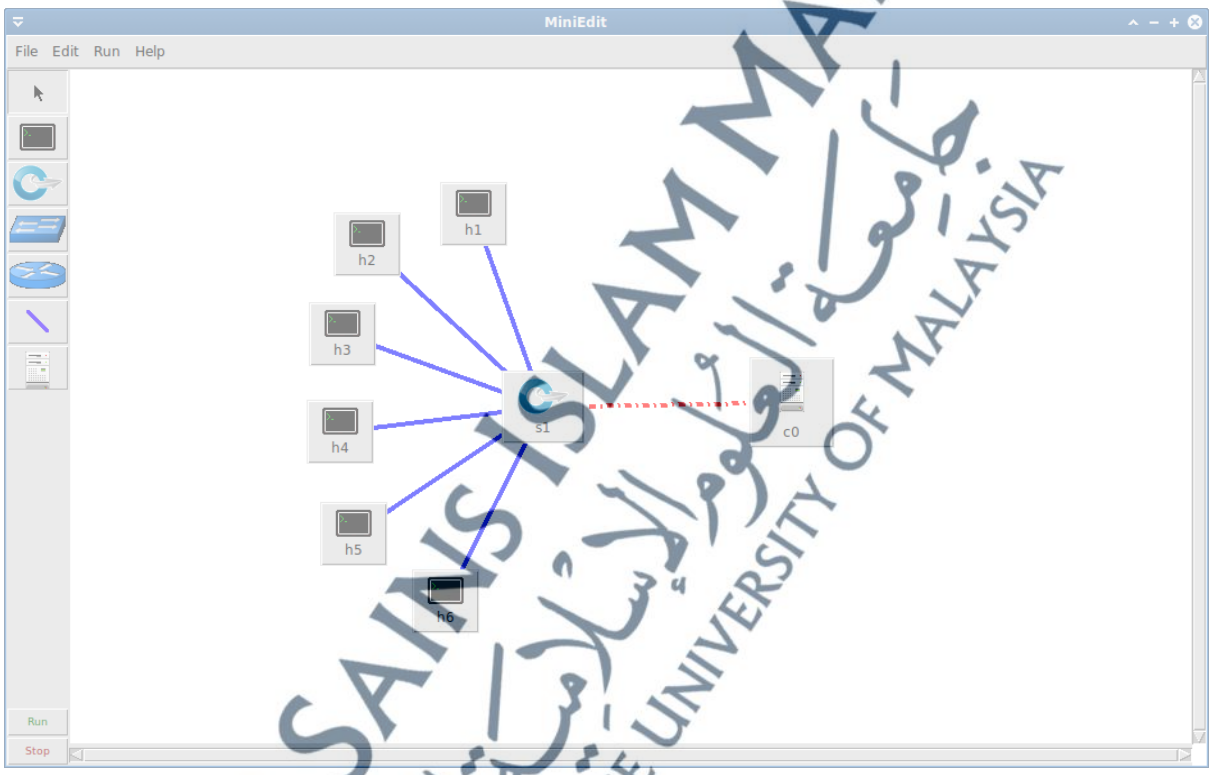

Figure 3.3: Simulation Topology

C0 and S1 denote the controller and the OpenFlow switch respectively. Open Virtual Switch (OVS) was used for network switches. OVS is a software switch that runs both on hardware and software. In Mininet, IP addresses are assigned incrementally for all hosts from 10.0.0.1 onward. Some of the six hosts are used to generate the normal traffic and others to generate the attack traffic. Different Python scripts are used for generating DDoS attack traffic as well as normal traffic.

3.3 Implementation of the Scheme Design

3.3.1 Experimental Hardware and Software

The experiments were conducted on a HP PC with Intel(R) core(TM) i5-3470 processor 3.20 GHz and 4GB of RAM. The operating system is Linux Ubuntu 14.04 installed on a VirtualBox virtual machine image and Mininet version 2.2.1 was run native on Linux.

Python was used to build and implement the scheme presented in this research. We used VIM version 7.4.52 as a text editor where the Python code was written. Microsoft Excel was used to plot all evaluated performance graphs in this research.

3.4 Testing and Evaluating the Implemented Scheme

In order to carry out the evaluation phase, we perform four different experimental scenarios each of which use a different DDoS attack and different test cases. The first experiment is a UPD flooding attack; the second experiment is a low-rate SYN attack and the third experiment is mixture traffic of all mentioned attacks and normal traffic. The last experiment is conducted to confirm the validity of the results by comparing the results obtained by the scheme to its results in the third experiment. The test cases used in the first and second experiments are normal traffic, attack traffic and a mixture of normal and attack traffic. The mixture of normal and attack traffic is the only test case in the third and fourth experiments. In the first experiment, the second test case, attack traffic is generated in two different packets scales namely a low packets scale and a high packets scale to evaluate the scheme in terms of detecting the change in the volume of the attack under the UPD flooding

attack. In the second experiment, the second test case, constant low rate traffic is generated to evaluate the scheme in short and long terms under low rate SYN attack.

Before we conduct the four experiments, we need to verify and validate the simulation topology, network emulator and, select the performance metrics.

3.4.1 Verification of the Network Emulator

Mininet emulator provides testing environment tools to allow both verification and validation. For this purpose, the simulation topology is created using Mininet Command Line Interface (CLI) from a Mininet terminal as shown in Figure 3.4. This figure shows that the build of the topology is working properly. Also, we can see that the programming code of creating the topology works as advertised.

Figure 3.4: Creating Simulation Topology using Mininet CLI

After running the Mininet emulator, many testing tools can be used to validate the installed network emulator.

3.4.2 Validation of the Network Emulator

Mininet emulator has many different testing tools used for different purposes such as testing connectivity between the added hosts, checking the availability of the added links between the hosts and others. Tools and results of using these tools appear in Figure 3.5. All the tests were successfully ran and no errors appeared.

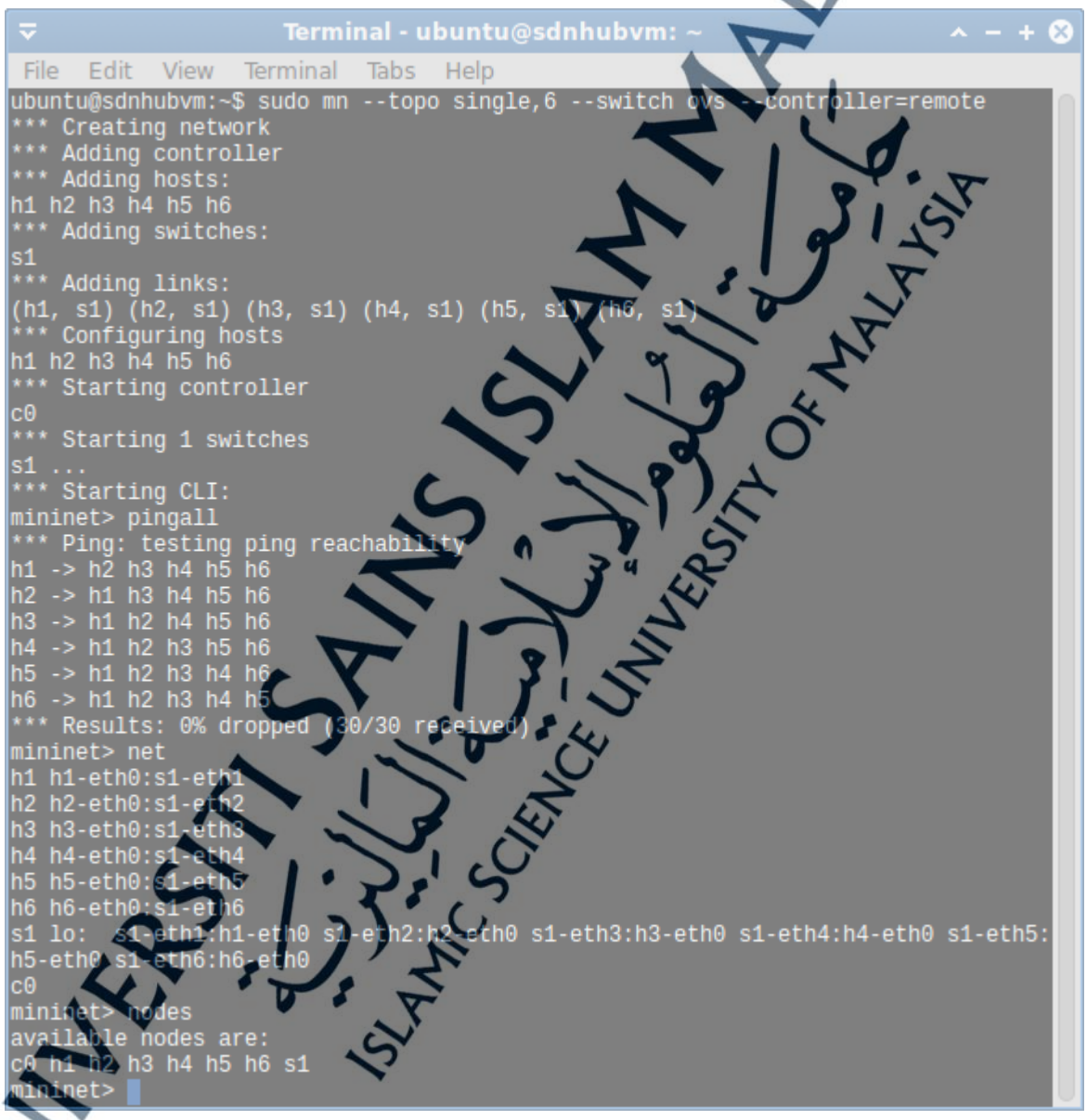

Figure 3.5: Results of Using Testing Tools

In this research, we used the Mininet verification and validation tools to confirm the correctness of the simulation topology. Then, we validate the proposed

scheme by comparing its results to the results of previous works in terms of overhead, accuracy and false alarms.

3.4.3 Performance Metrics

As a step toward evaluating our findings in this research, we experimented and compared the performance of the proposed scheme to the previous research studies. Some network performance metrics were selected to do the comparison. In particular, we concentrate on the performance metrics overhead and accuracy to confirm the performance enhancement achieved by the new scheme. The percentage of the false alarms produced by the proposed scheme is also included in the comparison. These performance metrics are measured on every third test case in every experiment because the third test case is mixture traffic.

3.4.3.1 Overhead (CPU usage)

Amidst the intention of designing a solution that has minimal effects on the controller in terms of resource usage, the proposed scheme has designed.

To evaluate the effect of the scheme on the CPU usage, we left all experiments run with the processes that currently run on Linux PC (using Ubuntu 14.04). We left the system monitoring application of Linux (htop) run in each one of the mentioned test cases. A screen of the htop application is captured to show the CPU usage.

3.4.3.2 Accuracy

In DDoS detection, the accuracy appears by dropping the attack packets after differentiating them from legitimates, which is resulting in decreasing the False Positives (false alarms). False Alarms means the attack packets classified as

legitimates. In this research, we compute the accuracy based on the equation shown in (3.1).

$$\text{Accuracy} = \frac{\text{Detected DDoS attack packets}}{\text{Total number of packets}} \times 100 \quad \text{.....(3.1)}$$

Where the number of the DDoS attack packets detected by the scheme is obtained by subtracting the total number of captured packets generated in the experiment (appeared in the source IP list) from the number of the attack packets that have been appeared on the POX terminal. Then we divide the number of detected DDoS attack packets by the total number of packets (appeared in the source IP list) multiplied in one hundred.

For more confidence, the scheme produces a list of the attack source IP addresses to make sure that there is no appearance of the source IP address that generate normal traffic, especially when the traffic generated in the network is mixture traffic. The total number of packets that currently exist in the switch after dropping the detected packets and, the list of the attack source IP addresses that appear on the POX terminal, all of them appear each time the number of packets exceeds the first threshold within every five seconds. The number of the detected and terminated flows appears on POX terminal each time number of flows exceeds the second threshold.

Due to not allowing the number of flows to be sharply increased in the switch, the scheme shows only the number of the terminated flows that equal to the limit stated in the threshold not the total number of flows that have been terminated. Also, due to knowing that a single packet can create a separate flow, the number of the DDoS attack flows detected by the scheme is obtained by adding the number of

attack packets detected in the experiment (appeared in the source IP list) to the total number of legitimate packets that have been appeared in the same list and subtracting it from the total number of packets. Then, we can get the number of attack flows. Next, we add the number of attack flows to the number of attack packets to get the total number of attack packets that sent by the flooding DDoS attack or low-rate DDoS attack. Then, use the equation in (3.1) to compute the accuracy.

Appearance of the source IP address that generates normal traffic in the list will be used as an indication of a false alarm. We compute the false alarms based on the equation shown in 2.

$$FA=1–TN \quad \text{..... (3.2)}$$

where TN (True Negatives) are legitimate packets classified as legitimates. The True Negatives computed based on the equation shown in 3.

$$TN = \frac{TN}{TN + FP} \quad \text{..... (3.3)}$$

where FP (False Positives), are legitimate packets classified as attack packets.

3.5 Summary

This chapter described the methodology of evaluating the new scheme and validating its results. This chapter introduced a brief background study about validation techniques used for DDoS research. Then, we elaborated more about the network simulation approach selected to evaluate the performance of the new scheme. Moreover, we discussed the selected simulator and emulator. Also, selecting the

appropriate programming language as well as the appropriate parameters for the detection and mitigation functions were discussed. The reason why these elements were selected is also described. In addition, the experimental design where the verification and validation of the simulation topology and the validation of the network emulator were described. Also, we discussed the experimental hardware and software that the new scheme is implemented on. Finally, testing and evaluating the performance of the new scheme described in details. The performance of the new scheme is evaluated based on the performance metrics namely overhead (CPU usage) and accuracy. False alarms also considered in the evaluation. Then, we compare our scheme's results with previous works to validate our findings and to confirm the enhancement achieved.