DATA PLANE-BASED DEFENSE SYSTEM AGAINST DDOS ATTACKS FOR SOFTWARE DEFINED NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

AHMET GÖZÜTOK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

MARCH 2018

Approval of the thesis:

DATA PLANE-BASED DEFENSE SYSTEM AGAINST DDOS ATTACKS FOR SOFTWARE DEFINED NETWORKS

submitted by **AHMET GOZUTOK** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Tolga Çiloğlu
Head of Department, **Electrical and Electronics Engineering** _____

Assoc. Prof. Dr. Cüneyt Fehmi Bazlamaçcı
Supervisor, **Electrical and Electronics Engineering Dept., METU** _____

Examining Committee Members:

Prof. Dr. Şenan Ece Güran Schmidt
Electrical and Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Cüneyt Fehmi Bazlamaçcı
Electrical and Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Ertan Onur
Computer Engineering Dept., METU _____

Assoc. Prof. Dr. Pekin Erhan Eren
Information Systems Dept., METU _____

Prof. Dr. Bülent Tavlı
Electrical and Electronics Engineering Dept., TOBB-ETU _____

Date: 23/03/2018

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: Ahmet GÖZÜTOK

Signature :

ABSTRACT

DATA PLANE-BASED DEFENSE SYSTEM AGAINST DDOS ATTACKS FOR SOFTWARE DEFINED NETWORKS

Gözütok, Ahmet

M.S., Department of Electrical and Electronics Engineering

Supervisor: Assoc. Prof. Dr. Cüneyt Fehmi Bazlamaçcı

March 2018, 97 pages

Software Defined Network (SDN) is a new networking architecture. It offers promising advances and provides remarkable solutions to certain challenges in this area, yet it is still vulnerable to Distributed Denial of Service (DDoS) attacks. DDoS attacks cause devastating impacts on the SDN architecture, which may lead to failure of an entire SDN network. There is no generally accepted network defense system against these attacks for SDN architecture; in addition, there are many unresolved problems in this area. This thesis provides the MiddleModule system, which is a Network/Transport-Level DDoS attack detection and prevention system framework designed for SDN architecture. The MiddleModule system proposes a data plane-based DDoS defense system, which means this system suggests deploying the monitoring, detection and the prevention capabilities into the data plane devices, namely OpenFlow switches. In addition, the thesis states several requirements that a data plane-based defense system

should satisfy and provides several attack detection algorithms against various Network/Transport-Level DDoS attack types. In the scope of this thesis, an extensive evaluation is performed on the proposed framework and on the detection algorithms, using different evaluation scenarios. The evaluation results are compared with the similar studies in the literature. Moreover, a detailed literature analysis is provided in this thesis, by explaining and classifying the related studies.

Keywords: SDN, Software-Defined Networks, DDoS, Distributed Denial of Service, network security

ÖZ

YAZILIM TANIMLI AĞLAR İÇİN DAĞITILMIŞ HİZMET REDDİ SALDIRILARINA KARŞI VERİ KATMANI TABANLI SAVUNMA SİSTEMİ

Gözütok, Ahmet

Yüksek Lisans. Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Cüneyt Fehmi Bazlamaçcı

Mart 2018, 97 sayfa

Yazılım Tanımlı Ağ (YTA) yeni bir ağ mimarisidir. Bu mimari, umut verici gelişmeler sunmakta ve bu alandaki birtakım zorluklara karşı dikkat çekici çözümler sağlamaktadır ancak Dağıtılmış Hizmet Reddi (DHR) saldırılarına karşı halen savunmasızdır. DHR saldırılarının, YTA mimarisi üzerinde yıkıcı etkileri vardır ve bu etkiler, bir YTA ağının tamamen çökmesine sebep olabilir. YTA mimarisine yönelik bu saldırılara karşı genel olarak kabul gören bir ağ savunma sistemi bulunmamaktadır. Hatta bu alandaki birçok problem hala çözülememiştir. Bu tez, YTA mimarisi için tasarlanmış bir Ağ/İletim Seviyesi DHR saldırı tespit ve önleme sistemi olan MiddleModule sistemini sunmaktadır. MiddleModule sistemi, veri katmanı tabanlı bir DHR savunma sistemi önermektedir, yani bu sistemde, veri izleme, saldırı tespiti ve saldırı önleme yetenekleri, veri katmanında bulunan cihazlara kazandırılmaktadır. Bu tez ayrıca, veri katmanı tabanlı bir ağ güvenlik sisteminin sahip olması gereken birtakım özellikleri belirtmekte ve

çeşitli Ağ/İletim Seviyesi DHR saldırı türlerine karşı bazı saldırı tespit algoritmaları sunmaktadır. Bu tez dahilinde, önerilen savunma mimarisi ve saldırı algılama algoritmaları, çeşitli test senaryolarıyla, kapsamlı bir değerlendirmeye tabi tutulmuştur. Değerlendirme sonuçları, literatürdeki benzer çalışmalarla karşılaştırılmıştır. Ayrıca, bu tezde, konu ile ilgili çalışmalar, açıklanarak ve sınıflandırılarak, detaylı bir literatür analizi de sunulmaktadır.

Anahtar Kelimeler: YTA, Yazılım Tanımlı Ağ, DHR, Dağıtılmış Hizmet Reddi, ağ güvenliği

To My Lovely Wife Hilal and My Parents

ACKNOWLEDGEMENTS

I want to express my sincere gratitude to my supervisor Assoc. Prof. Dr. Cüneyt Fehmi Bazlamaçcı for giving me the honor of working with him. Without his constant support and guidance, it was impossible to perform this work.

In addition to my advisor, I am gratefully indebted to the rest of my thesis committee: Prof. Dr. Şenan Ece Güran Schmidt, Assoc. Prof. Dr. Ertan Onur, Assoc. Prof. Dr. Pekin Erhan Eren, and Prof. Dr. Bülent Tavlı for their insightful comments and contributions to my thesis.

I would also like to thank to my company, ASELSAN for supporting me.

Last but not the least, I want to thank my wife, Hilal, for being in my life, and of course my parents for their spiritual supports.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Software Defined Network (SDN) is a new networking paradigm and architecture. It offers promising advances and attracts the attention of both the academia and industry, yet it is vulnerable to Distributed Denial of Service (DDoS) attacks [1]. DDoS attacks have always been a major problem in networking area. However, for SDN networks, the effects of DDoS attacks are more devastating than traditional networks. In fact, SDN may become the real target of these attacks and an entire SDN network may become unavailable to legitimate users because of these attacks [1]. Defense methods against DDoS for SDN is still an unresolved area, and there is no dominant solution accepted for this problem [2]. In this thesis, we suggest that a data plane-based DDoS detection and mitigation technique, which satisfies several requirements, would provide an effective defense mechanism against DDoS attacks for SDN environment. To substantiate our suggestion, we designed the MiddleModule system - a data plane-based detection and mitigation framework - and proposed several detection algorithms along with it and evaluated the proposed system extensively.

1.1 Overview

In networking, a malicious user may send malformed messages to a victim server to disrupt its accessibility by legitimate users, which is called as Denial of Service (DoS) attack. When a malicious host uses some bots or reflection technique to generate the DoS attack, it becomes a Distributed DoS (DDoS) attack.[3] A malicious user can easily generate a large amount of malicious traffic by using a DDoS attack, which threats both the victim servers and the underlying network. Due to advancements in networking area and the Malware-as-a-Service (MaaS) approach, the volume of DDoS attacks are increasing significantly [1]. A DDoS attack in the

order of Tbps has been recorded in 2016 [4].

Software-Defined Networking (SDN) and DDoS attacks, specifically the Network/Transport-Level DDoS attacks, have a contradictory relationship. SDN suggests a layered architecture, and in SDN, network is separated into application-plane, control-plane and data-plane. SDN provides several advantages to detect a DDoS attack towards a victim host by using this layered approach, such as centralized-control, network-wide view and dynamical forwarding rules. However, despite these features, SDN itself may become the target of a DDoS attack. The layers of SDN or the communication channels between these layers can be the main targets and the SDN nodes may become inaccessible to legitimate users under a DDoS attack. Although there exist several studies in this area, there is no dominant solution for these problem; in addition, there remains many unresolved problems in this area.

1.2 Aim of the Thesis

Although SDN brings several great features in networking, the use of SDN is still very limited because of the vulnerabilities of SDN to DDoS Attacks [1]. In this thesis, we aim to clarify the main problems of SDN, which may lead to failure under DDoS attacks. In addition, we focus on providing a scalable and comprehensive defense mechanism which can effectively detect DDoS attacks with low system performance degradation and with high defense strength performance. We also aim to explain the well-known defense methods suggested in the literature so far and classify them.

In this thesis, we suggest to use a data plane-based DDoS detection and mitigation technique. We first state several requirements that a data plane-based method should satisfy to obtain the best performance. By considering these requirements, we provide the MiddleModule framework, where each edge-switch of an SDN network collects statistics and applies statistical-based detection algorithms on incoming packets. If any malicious activity is detected, the appropriate mitigation technique is applied by that edge-switch. To provide such functionality effectively, we use

multiple location defensive approach, which means, different detection functions are performed at different locations of the network. With the proposed defense system, we do not send the malicious traffic to the controller because we aim to avoid controller overloading and switch-controller communication channel congestion problems but instead we detect and react the malicious traffic at the closest switch to the source of the malicious activity. We perform functional analysis and performance evaluations for the proposed method and test results are compared with similar studies found in the literature. Our test and comparison results suggest that the proposed system performs well.

1.3 Contributions of the Thesis

The contributions of this thesis can be summarized as follows;

1. We provide a comprehensive explanation of the effects of Network/Transport-Level DDoS attacks on SDN. We explain and compare the existing defense methods.

2. We state some basic requirements that a data plane-based DDoS defense system for SDN should satisfy, such as having lightweight algorithms and using multiple-location defense approach.

3. By using the suggested requirements, we provide a data plane-based DDoS defense framework, the MiddleModule framework, which brings monitoring, detection and prevention capabilities to the edge-switches. We also provide several DDoS detection algorithms expected to be effective against the most common examples of each Network/Transport-Level attack type and implement these algorithms within the MiddleModule framework.

4. An extensive evaluation is performed to demonstrate that the proposed method is capable of effectively defending an SDN network against Network/Transport-Level DDoS attacks. The proposed method is simulated on OMNET 4.2++ environment to perform a detailed functional analysis under different

test scenarios. In addition to simulation, the proposed method is also emulated on the Mininet environment to analyze the performance of the method and to carry-out the proof-of-work. By using the test results, the proposed method is compared with the similar studies found in the literature and the advantages and disadvantages are discussed throughout the thesis.

1.4 Thesis Outline

The thesis is organized as follows;

In Chapter 2, the background information about both the SDN and DDoS concepts are given. The contradictory relationship between SDN and DDoS is discussed. The well-known DDoS mitigation methods for SDN in the literature are classified and explained.

In Chapter 3, the proposed framework, the MiddleModule framework, is explained and several requirements that a data plane-based DDoS mitigator should satisfy are suggested with their justifications. The sub-blocks of the proposed framework are defined and explained in detail.

In Chapter 4, several DDoS detection algorithms, designed to operate within the MiddleModule system, are proposed. The way these algorithms are used within the framework is described. The pseudo-codes of these algorithms are given, and the capabilities of these algorithms are discussed.

In Chapter 5, the evaluation phase is explained by describing our test platforms, test scenarios and test results. A detailed functional analysis is performed in OMNET 4.2++ simulation environment and an extensive performance evaluation is performed in Mininet emulation environment. The details of these processes, and the results of the associated tests are explained in this chapter.

Chapter 6 concludes the study stating also some potential future directions.

CHAPTER 2

BACKGROUND AND LITERATURE OVERVIEW

2.1 BACKGROUND INFORMATION

2.1.1 Software-Defined Network

Software-Defined Network (SDN) is a new paradigm in networking area. SDN proposes a different network architecture than traditional network by decoupling the network into different planes, application-plane, control-plane and data-plane [1]. In traditional networks, the two-fundamental networking processes, routing and forwarding, are handled by routers. Routers are distributed across the network, and they operate in a stand-alone fashion by communicating with each other. On the other hand, as shown in ***Figure 2.1***, in SDN, these two processes are handled at different devices, which are deployed at different layers of network. The routing decisions are made by a logically centralized device, which is deployed at control-plane of the network, called as controller. The controller have a global network view and it runs the required routing algorithms, along with the other algorithms that belong to application-plane [5]. In a sense, control-plane represents the intelligent part of the network. On the other hand, the forwarding process is handled by logically and physically distributed devices, which are deployed at data-plane of the network, called as switches. The switches forward packets according to the forwarding rules received from the controller; therefore, the data-plane is the unintelligent part of the network [1]. Each switch communicates with the controller over a secure channel and receives the forwarding rules over that channel by using a well-defined protocol. In networking society, OpenFlow protocol is widely used as the switch-controller communication protocol [6].

In SDN architecture, all hosts are directly connected to the data-plane devices, as shown in ***Figure 2.1*** and data-plane devices (switches) are connected to each other to provide required connectivity between hosts. When a switch receives a packet from a host, it checks if there exists a matching flow entry in its flow table for the received packet. If there is a match, the switch forwards the packet to one of its ports according to the corresponding forwarding rule. If there is no match, then by using the switch-controller communication channel, the switch sends a *packet_in* message to the controller with the header portion of the received packet while the data portion of the packet is stored in the switch buffer. SDN Controller processes the *packet_in* query by using its routing algorithms and sends the corresponding forwarding rule to the switch. Switch stores the forwarding rule to its forwarding table, dequeues the stored data portion of the packet, and forwards the packet to its corresponding port. This packet will be forwarded from switch to switch lying in the corresponding routing path and finally the packet will arrive at the destination host [1].

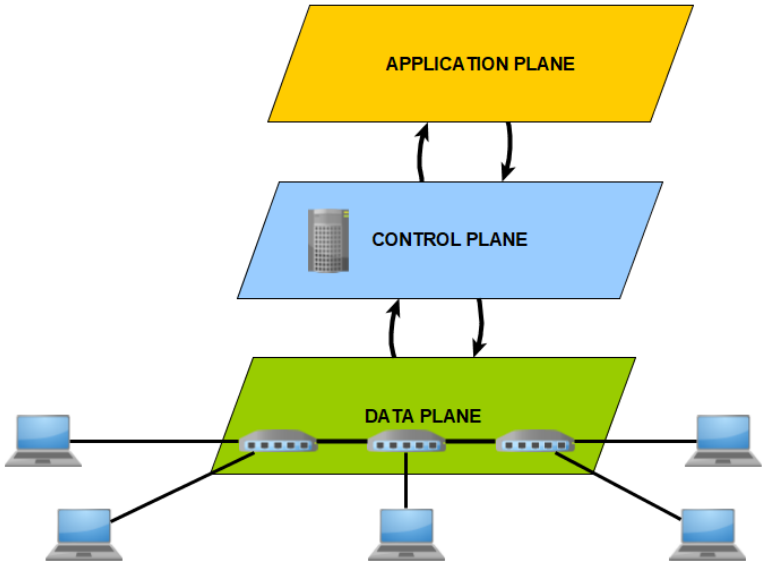


Figure 2.1: The basic Model of SDN

The separation of data plane and control plane, which is the main idea behind the SDN, has a longer history than SDN architecture [5]. In order to improve the telephone networks, early attempts of data/control plane separation is applied around

1980s and 1990s [5]. In addition, active networks or programmable data plane concepts are also not very new in networking area. Programmable switches were discussed around 1990s and several methods to realize this concept were suggested such as downloading some instruction sets into switches and altering the packet forwarding rules of the switches [5]. SDN, on the other hand, represents a more compact and protocol-based architecture. In fact, the origins of SDN can be considered as the studies and ideas around OpenFlow protocol [5]. There are several advantages of SDN/OpenFlow, such as providing centralized management, adjusting network-wide traffic according to changing needs, managing network resources via automated and dynamic SDN programs, and managing the network independent of vendors [7]. However, the real motivation behind the SDN/OpenFlow architecture is isolation and virtualization [8]. SDN/OpenFlow allows network engineers to operate different network routing rules and to provide different connection trees on the same network for different isolated traffics belonging to different users simultaneously. SDN has too many advantages for the academic use and for the business use [8]. Several companies including Google, Akamai, Cisco and Microsoft are members of Open Networking Foundation (ONF) with the aim of advancing SDN using open standards. Recently, Google built an SDN-based interconnection network to connect its data centers around the globe. Although SDN draws attention in networking society, industry experts suggest that security issues of SDN are still immature and further investigation in security of SDN is required [5].

2.1.2 Distributed Denial of Service

Denial of Service (DoS) attacks are aimed to disrupt the accessibility of a victim server by sending malformed messages. The attacker may have many incentives to attack a server such as financial gain, revenge, ideological belief, intellectual challenge or cyberwarfare [3]. When the malicious activity is supported by more than one malicious user, the DoS attack becomes Distributed DoS (DDOS). DDoS attacks are much more challenging than DoS attacks, because by using a DDoS attack, malicious users can easily generate a large amount of malicious traffic. Under such attack, the victim server becomes inaccessible to legitimate users. In addition to the

victim server, the underlying network also suffers from such attacks. The underlying network will be congested with too many packets and the network devices may fail to provide some services to legitimate users due to buffer overflows and excessive packet drops. DoS attack has been a problem for computer networking for a long time. The first known DoS attack is encountered at early 1980s [3]. As computer networks get larger and as Internet becomes popular around the world, DDoS attacks started to appear, and the first DDoS attack was recorded in 1999 [3]. As networks became wider, DDoS attacks became more complex and the capabilities of these attacks has improved. In October 2002, a DDoS flooding attack against 9 DNS root servers was executed and consequently 9 out of 13 DNS root servers were inaccessible to entire Internet users around the world. Since Internet technologies are more advanced now, several attacking tools are developed; in addition, Malware-as-a-Service (MaaS) concept is also introduced. Today, a malicious user may access a botnet army with a low cost (e.g. a botnet army with 10.000 computers for $1.000) and generate great amount of malicious traffic easily [1]. Therefore, making a DDoS attack has become much easier and the size of the DDoS attacks has increased nearly exponentially. The size of the largest reported DDoS attack in 2010 is approximately 100Gbps, while in 2013 it is nearly 300 Gbps [1]. Cloud computing getting more popular every year, DDoS attacks became immense and more flexible. A DDoS attack in the order of Tbps is recorded in 2016 [4].

DDoS attacks are classified into two categories; Network/Transport-Level DDoS Flooding Attacks and Application-Level DDoS Flooding Attacks, as shown in ***Figure 2.2*** [1]. The former is focused on generating DDoS attacks by using mostly TCP, UDP, ICMP and DNS protocol packets. In this attack type, the aim is exhausting the bandwidth of the victim by sending too many packets to the victim server; consequently, the connectivity of the victim server will be disrupted, and legitimate users cannot access the victim server [3]. These are volumetric attacks. The latter is focused on generating application level messages and the aim is exhausting the resources (e.g. CPU, memory, sockets and I/O bandwidth) of the victim; consequently, disrupting the services of the victim server. Application-Level DDoS Flooding Attacks consume less bandwidth when compared to the

Network/Transport-Level DDoS Flooding Attacks, and they are stealthier and more sophisticated [3]. Although Application-Level DDoS Flooding Attacks are problematic for victim server, Network/Transport-Level DDoS Flooding Attacks cause severe problems for both the victim server and the underlying network. In addition, Network/Transport-Level DDoS Flooding Attacks cause devastating impacts on SDN network [1]. Therefore, in this thesis, we prefer to focus on providing a defense mechanism against Network/Transport-Level DDoS Attacks. Network/Transport-Level DDoS Attacks are classified into four categories in the literature, as shown in ***Figure 2.2*** [3]:

1. **Flooding Attacks:** This is the basic Network/Transport-Level DDoS Attack type and it aims to exhaust the victim server's network bandwidth and disrupt legitimate connection to this server. The examples for flooding attacks are spoofed/non-spoofed UDP Flood and ICMP Flood.
2. **Protocol Exploitation Attacks:** In this attack type, the attacker aims to consume the resources of the victim server by exploiting some features of victim's connection protocol or by exploiting implementation bugs. The well-known example of this attack type is spoofed/non-spoofed SYN Flood attack.
3. **Reflection-Based Attacks:** Malicious users send malformed requests to reflector users instead of directly sending these to the victim; in this case, a reflector user is a third host in the network other than the attacker and the victim. In the malformed requests, the source address part is changed with the address of the victim. Therefore, the replies from the reflectors will be sent to the victim, and the bandwidth and the resources of the victim will be exhausted. An organized and distributed flooding attack, such as Smurf attack, can be considered as an example of this attack type.
4. **Amplification-Based Attacks:** Attackers may intent to increase the effects of their attacks by amplifying the attack towards the victim. This attack generally used with Reflection-Based Attacks. By using an Amplification-Based Attack, the attacker consumes much more bandwidth of the victim server than its own bandwidth to generate such attack. One of the most common usage is making the reflectors to generate broadcast messages, as in the case of Smurf attack.

Source address spoofing attack is not classified as a different attack type, instead it is discussed under other attack types (e.g. spoofed/non-spoofed flooding attacks) in the literature. However, this attack itself can be problematic for SDN networks. To handle this attack in detail and to provide a comprehensive defense mechanism against it, and to test it carefully, we consider this attack as a different type and treat it as the fifth Network/Transport-Level Attack Type.

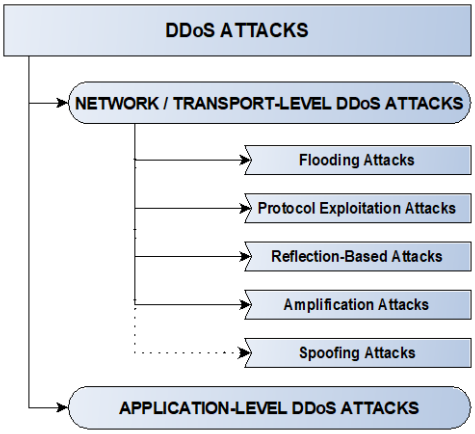


Figure 2.2: DDoS Attack Types

2.1.3 Contradictory Relationship between SDN and DDoS

DDoS attacks and SDN architecture has a contradictory relationship. SDN has very powerful capabilities to detect and prevent a DDoS attack through a victim server; however, SDN itself may become the victim of a DDoS attack. In order to defend hosts against DDoS attacks, SDN has several features, such as i) logically centralized controller and global network view ii) programmability of network elements iii) dynamic forwarding rules iv) distributed nodes across the network v) testing a defense system easily [1]. Despite these great features of SDN that can be used to defend hosts against DDoS attacks, SDN itself may become the target of these attacks since SDN is designed to serve pure-minded hosts. A malicious DDoS activity, can disrupt the services of SDN nodes that may lead to failure of an entire network. Spoofed or non-spoofed, originated by several zombie hosts or bots, large

number of packets may cause severe problems [9]. Some of these problems are explained as follows:

1. **Switch Buffer and Flow Table Overflow:** When an OpenFlow switch receives a packet whose flow ID is not in the flow table of the switch; the switch stores the data portion of the packet to its buffer and sends the header portion to the controller with a *packet_in* request message. Controller processes the incoming request and sends the appropriate flow rule to the switch, which is stored in the flow table of the switch. If a switch receives too many packets from hosts with unknown flow IDs, the switch buffer overflows, and when a packet with unknown flow ID is received, instead of storing its data portion, the switch sends the entire packet to the controller. During a DDoS attack, buffers of switches can easily be overloaded. In addition, the redundant flow rules received from the controller are also stored in switch flow tables. Since each switch has a limited size flow table, when too many redundant flow rules are received, legitimate ones may be dropped from the flow table and forwarding speed of legitimate traffic may dramatically decrease [10].
2. **Switch-Controller Channel Bandwidth Congestion:** When switches send too many *packet_in* messages in a short time interval, the switch-controller communication channel becomes congested. In addition, following a switch buffer overflow, this communication channel congests more easily, and delays or packet drops occur on this channel, which carries all OpenFlow traffic between switches and controllers including periodic and sporadic messages. Therefore, when such a problem occurs, all switches sharing such a communication channel will suffer and forwarding performance of the associated switches degrades notably [10].
3. **Controller Buffer Overflow:** When controller receives a *packet_in* request, it processes the request and finds the appropriate forwarding rules. Computing such actions for each request consumes controller resources, such as memory and CPU. Although a controller is a powerful device, still it has limited amount of resources. During a DDoS attack, a large number of redundant *packet_in* requests may saturate the controller's resources; therefore, the controller starts to drop the legitimate *packet_in* requests or delay them. Therefore, in SDN, single point of failure problem arises under such attacks due to having a logically centralized controller. When the

controller drops or delays legitimate *packet_in* requests, the entire network fails to operate correctly. The controller may become unreachable to legitimate users. This problem occurs even if there is a backup controller or the controller is physically distributed because of the logically centralization of the controller [10].

To sum up, Software Defined Networking architecture has several major problems, which makes the SDN vulnerable to Distributed Denial of Service attacks and these issues arise because of having the following three main characteristics: i) SDN trusts all connected hosts ii) logically centralized control plane of the SDN has single point of failure problem iii) SDN has limited size buffers and limited size switch flow tables [11]. Hence, any comprehensive DDoS defense mechanism for SDN should consider these.

2.2 LITERATURE REVIEW

The methods proposed in the literature, containing both SDN and DDoS, have mainly two different focusses, designing an SDN-based DDoS detection and mitigation technique for traditional networks or designing a DDoS detection and mitigation technique for SDN itself [2].

SDN is capable of providing flexible and powerful analysis techniques, which could help in mitigating DDoS attacks in traditional networks. Many studies are proposed for that purpose; however, in those studies the fact that SDN itself may become the real target of DDoS attacks is ignored, which causes a serious security gap. On the other hand, to make SDN itself immune to DDoS attacks, several methods are also suggested. The studies that consider defending traditional networks by using SDN approach are not within the scope of this thesis; therefore, they are not explained in this chapter. Instead, the studies that concentrate on improving the SDN architecture against DDoS attack are considered.

The existing methods can be classified into three categories, control plane-based methods, data plane-based methods and hybrid ones. In control plane-based methods, the detection operations are handled at the controller device by running a defense

application on it. On the other hand, in data plane-based methods, the attack detection and mitigation operations are handled at the data plane to avoid single point of failure problem by not sending malicious data to the controller for processing. Although the present thesis proposes a data plane-based defense mechanism, all three categories are investigated in detail and a brief survey is presented in the following sections.

2.2.1 Control Plane-Based DDoS Defense Methods

When a defense method has DDoS detection and mitigation module, or algorithm, running on the centralized controller, or any other centralized device, to provide a network-wide defense mechanism, it is called as Control plane-based DDoS Defense. These methods are popular because they can easily be designed and implemented in an SDN network. SDN centralized controller attracts also the attention of many researchers. Although data monitoring techniques or detection approaches differ among such studies, the framework used is generally common.

In [9], Mousavi *et al.* proposes using the controller to collect statistics and to detect DDoS attacks. For that purpose, an entropy-based application, which checks the variations of entropy of each destination IP address, runs on the controller device. If packets in the network are destined to all destinations homogeneously, then the likelihood of each destination to receive a packet will be close to each other, and the entropy will be high. On the other hand, if packets are destined to only one destination, then the likelihood of this destination to receive a packet will be high while others low, and the entropy value will be smaller. Controller calculates the entropy and compares it to a threshold. If the entropy is lower than a threshold, the controller concludes that there is a DDoS attack and implements mitigation rules to drop all packets destined to the victim server. Other similar works such as [12] and [13] exist, which propose centralized defense systems by using entropy based detection algorithms running on the controller.

Several methods have a similar framework as in [9], yet they differ in their detection algorithms. In [14], the statistics, including the number of connections and the

number of packets per connection for each flow, are collected by the controller and the statistics are compared with thresholds to make a decision. Several security applications such as firewalls and network anomaly detectors are implemented on the controller to detect any malicious activity in [15], [16], [17], [18], [19] and [20]. In [21], a DDoS mitigation technique for SDN-based cloud networks is suggested. The proposed method uses ALTO server to obtain a bird's eye view of the entire cloud network and the security algorithms run on the controllers of the SDN networks.

In [22], [23], [24], and [25] various Machine Learning-based detection algorithms are provided to avoid DDoS attacks. Machine Learning algorithms are implemented at the application layer, input features are collected from the network, and by using these features, malicious activities are detected. In [26] and [27], a high-level DDoS detection application is proposed using SOM detection algorithm that uses statistics collected by OpenFlow switches. In [27], an efficient resource utilization monitoring technique is also suggested for switches.

Several other methods use a centralized DDoS detection device other than the controller itself, such as [28]. These methods avoid overloading the controller with additional DDoS detection algorithm load; however, these devices suffer from communication channel congestion problem and the associated device resources can get exhausted. Hence, these methods face single point of failure problem under an organized DDoS attack.

Different statistics collection techniques are proposed, some of them suggest using a third-party device for this purpose [29]; while others point out that OpenFlow switch capabilities and OpenFlow protocol itself is enough for statistics collection [30].

Some methods concentrate only one of the main problems caused by DDoS attacks. Although these methods propose promising solutions to its associated specific problem, they generally have negative impacts on other problems. In [31], to avoid OpenFlow switch flow table overloading problem, QoS-aware load balancing strategy between switches is suggested. Centralized controller continuously checks the status of each switch and if the flow table of a switch is full, then the traffic

destined to that switch is re-distributed to other switches making the switch not experience flow table overloading. However, this method considers only the flow table overloading problem and to solve this problem, it puts an additional processing burden on the controller, which may exhaust controller resources even faster under a DDoS attack. In [32], to avoid the controller overloading problem, a processing queue scheduling algorithm for controller is suggested. The controller logically subdivides its queue into different logical queues, each one of them being reserved for different switches and the controller serves each logical queue in a round robin fashion, or any other way. Since this method only considers controller overloading, when a switch is under a DDoS attack, it will face the buffer overloading problems even faster; the legitimate traffic, passing through the associated switch, will experience large delays or they will not get any service at all.

Although the control plane-based methods seem convenient to use, they do not provide a comprehensive solution to the main problem of the SDN, i.e., single point of failure under DDoS attacks. In fact, adding DDoS detection/mitigation applications, in addition to standard controller applications, and analyzing each packet on controller may exhaust the resources of controller even faster. Adding an additional centralized detection device is also not a solution to the problem, because these devices inherit single point of failure problem from the controller. Furthermore, since malicious packets are still sent to the controller, switch-controller channel congestion problem also remains unsolved.

2.2.2 Hybrid DDoS Defense Methods

Hybrid DDoS defense mechanisms use both data plane devices and control plane devices during DDoS detection and mitigation operations. The packets are monitored at data plane devices and suspicious packets are found somehow in the data plane devices; then all the suspicious packets are directed towards the controller for further analysis. The main DDoS detection and prevention processes are handled by the controller.

In [33], a defense method for DDoS attacks in SDN-based cloud networks is proposed. In this method, SDN switches distinguish suspicious and nonsuspicious packets with a primitive analysis and then forward suspicious packets to the controller for further analysis and controller detects the attacks and decides appropriate mitigation rules. In [34], a multi-level DDoS detection method is proposed, where attack monitors and correlators are distributed across the network and placed at switches to detect any malicious activity. Monitors run Snort-based IDS and if any anomalous activity is detected, the anomalous packet is forwarded to the correlator that checks if the source address is spoofed or not. If so, these packets are directed to the controller for further analysis and also for the generation of a prevention rule. Therefore, under a volumetric DDoS attack, the controller device still gets overloaded; hence, the network is faced with the single point of failure problem. In [35], Network Function Virtualization (NFV) and SDN is combined for DDoS detection/mitigation purpose. By using NFV, specific monitoring functionalities are added to the data plane and a centralized orchestrator is used to coordinate these virtualized functions. Monitored data is analyzed at the controller and the required mitigation functions are implemented at the data plane in the form of virtual functions or forwarding rules.

Hybrid methods decrease the workload of the centralized controllers; however, in these methods, all the malicious traffic is still directed to the controller for detection and mitigation purposes. Although these methods improve the performance of SDN against controller overloading and communication channel congestion problems, they still cannot provide an adequate solution to these problems.

2.2.3 Data Plane-Based DDoS Defense Methods

Data plane-based DDoS defensive mechanisms seem more suitable for providing a comprehensive solution to all problems of DDoS attacks when compared with others. Adding intelligence to data plane devices helps to detect and prevent DDoS attacks at data plane and keep malicious flows within data plane. Therefore, controller, communication channel and the rest of the network are not prone to malicious traffic. In addition, an accurate response to a malicious activity can be generated relatively

quickly and effectively, because there is no propagation and queuing delay in a data plane-based method.

With data plane-based DDoS defense methods, the following question may rise; does adding such intelligence to data plane devices compromise the main paradigm of SDN. The main idea behind SDN architecture is using an intelligent device at control-plane to generate routing decisions and using several non-intelligent devices at data-plane to forward packets according to those decisions [5]. Although the common understanding of SDN is in this way, in the original architecture of SDN provided in [8], it is pointed that capabilities supported by a standard OpenFlow Switch are flexible and extensible. In fact, today, a standard OpenFlow Switch provides several additional functionalities other than packet forwarding [36]. All OpenFlow Switches have processors, several memory units and some software programs running on them. An OpenFlow Switch uses its processor to analyze incoming/outgoing packets for extracting a set of detailed statistics about them and to communicate with the controller [36]. Therefore, although, these devices are known as non-intelligent, they have some degree of intelligence, but they do not use their intelligence to generate routing decisions.

In [8], it is clearly stated that an OpenFlow Switch should include at least the following parts; *Flow Table, Secure Communication Channel* and *OpenFlow support.* In addition, several essential set of actions for OpenFlow Switches are provided, and the routing/forwarding operation for a standard SDN architecture is clearly explained. As long as routing/forwarding characteristics and other essential properties of SDN are not modified, extending the capabilities of an OpenFlow Switch by adding minor intelligence for defensive purposes does not compromise the main paradigm of SDN [2].

In [4] and [37], an in-switch processing method to detect and mitigate DDoS attacks is suggested. The monitored values are processed by an entropy-based algorithm running on the switches for the detection of malicious activity. If an attack is detected at the switch, the switch drops the flow and notifies the controller to designate the counter-measure. When a packet arrives at a switch, the entropy-based

algorithm calculates the entropy, checks if it is below a threshold, and if so, labels this packet as an attack. However, if more than one host connected to the same switch are under attack, this algorithm cannot detect the attack accurately.

AVANTGUARD, in [38], proposes a new framework for SDN to mitigate spoofed SYN Flooding Attacks. The SYN Flooding mitigation is achieved by Connection Migration block, which is implemented on OpenFlow switches. When a switch receives a "SYN" message, it is directed to the Connection Migration block. If the message has an unknown flow ID, the Connection Migration block automatically responds to that host with a "SYN+ACK" message. If the host sends an appropriate reply, then the switch receives a forwarding rule from controller for this connection and forwards all messages coming from the host to the destination. If the host does not reply to the "SYN+ACK" message, then the Connection Migration block drops that packet without keeping any state.

In [2] and [39], switches apply detection algorithms to incoming packets. Both methods propose detection algorithms inspired by the PacketScore [40] technique, which is designed for traditional networks. These algorithms include more than one sequential packet processing stages. These stages, basically, are monitoring, score calculation and mitigation; which are applied all to the incoming packets before forwarding them. Score calculation stage includes profile generation and profile updating phase, a probability-based score generation phase, and a threshold comparison phase. Although these studies provide promising performance results, detection algorithms implemented in the data plane devices seem a bit complex to be employed for data plane devices. Using such complex algorithms for all incoming packets before forwarding them may cause serious system performance degradation problems. In addition, in [39] an additional analysis stage, based on SYN cookie technique is proposed. The proposed analysis technique is very similar to the one proposed in AVANT-GUARD and adding this degrades system performance even more.

In [41] and [42], a collaborative DDoS detection method is proposed, where an Artificial Neural Network (ANN) is implemented on the SDN data plane and the

SDN switches are used as the ANN nodes. A 3-layered Radial Basis Function Neural Network is proposed for this purpose. All SDN switches are used as ANN input layer nodes; while, some of them are used as ANN hidden layer nodes and output layer nodes. The nodes receive three features: ratio of i) ICMP packet counts, ii) UDP packet counts iii) SYN packet counts, to all packet counts. After exchanging packets between neurons and processing the inputs, the output neurons decide if there is a DDoS attack in the network or not. The proposed systems do not provide any information about the source or the destination of the malicious activity.

Detecting the malicious activity at the data plane and mitigating the malicious packets without sending them to the controller device could provide a comprehensive solution for the problems of SDN caused by DDoS attacks. Since DDoS packets are not directed to the controller device, the controller is not overloaded, and the controller-switch communication channel is not congested. In addition, if the malicious activity is detected quickly enough in an efficient and effective way, the switch buffer and flow table overloading problems can also be overcome.

In this thesis, we highlight that a data plane-based mechanism is necessary for a comprehensive detection and mitigation system against Network/Transport-Level DDoS attacks; and we provide a data plane-based DDoS detection framework. We consider centralized controller as a vulnerability under DDoS attacks and instead of taking advantage of the centralized controller for defensive purposes, we aim to protect it by not directing the DDoS traffic to it. In this thesis, we take advantage of the data plane nodes of SDN, and their distributed and controllable nature. These nodes are distributed across the network and by using them, malicious packets can be processed very close to the sources and the victims of any malicious activity.

In this thesis, we suggest several requirements that a data plane-based defensive method should satisfy, and we provide a data plane-based defense system designed according to these requirements. In addition, we propose lightweight and scalable DDoS detection algorithms against all Network/Transport-Level DDoS attack types. Furthermore, we suggest using the multiple location defensive approach, which means we use different detection algorithms at different locations in the network.

CHAPTER 3

SYSTEM DESIGN

To provide a defense system against Network/Transport-Level DDoS attacks for SDN architecture, we present MiddleModule system, a data plane-based DDoS detection and prevention mechanism. In addition, we state several system requirements to create comprehensive data-plane defense mechanisms.

3.1 ARCHITECTURE OVERVIEW

The data plane of a standard SDN network includes OpenFlow switches connected to each other to provide the required connectivity in the network. As shown in ***Figure 3.1***, these switches can be classified as edge-switches and core-switches; where edge-switches are placed at the gateways of the network. On the other hand, the core switches are located at the center of the data plane network and they are connected to other switches in the network.

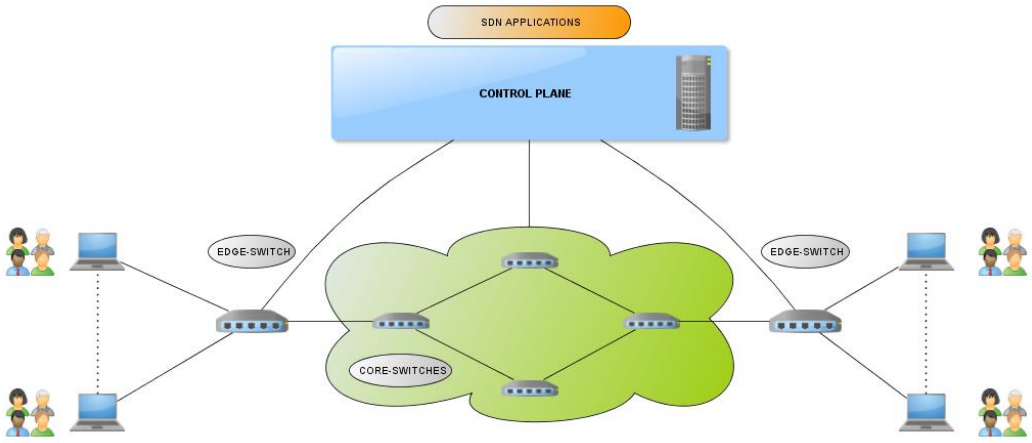


Figure 3.1: Standard SDN Network

In the MiddleModule system, the detection and mitigation functionalities are implemented in edge-switches, while core switches are used without any alteration, as shown in ***Figure 3.2***. In addition, an Orchestrator Module is implemented in the controller, which is used for communication of the controller with the MiddleModule blocks located within edge-switches.

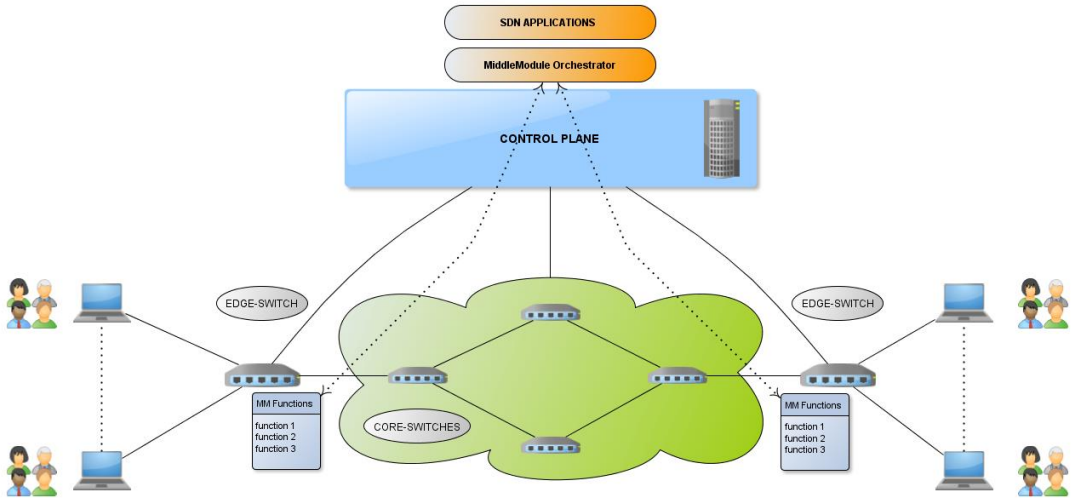


Figure 3.2: SDN Network with MiddleModule System

When any packet enters an SDN network that employs the MiddleModule, it is analyzed at the gateway, i.e., the edge-switch, of the network. Therefore, by utilizing a small amount of resources and simple detection functions, DDoS activities can be detected quickly and accurately at the gateway points. Using this method, malicious packets do not occupy resources of other network components.

There are two important points regarding the MiddleModule system. Firstly, although there is an Orchestrator Module in this system, running on top of the controller; the MiddleModule system is a data plane-based defense system; and the Orchestrator Module is not used for detection/mitigation purposes. The DDoS detection functions that are located in the edge-switches do not need any periodic messages or commands from the Orchestrator Module in order to operate correctly. MiddleModule functions operate in a stand-alone fashion. The Orchestrator Module provides the ability to the network operator to configure the MiddleModule functions as desired, or the ability to enable or disable them. In SDN, it is important for the

controller to be aware of any forwarding action applied by the switches in the network, including packet dropping rules. Therefore, when a MiddleModule function, located in an edge-switch, detects a malicious activity and decides to use a prevention rule, it informs the controller by using the Orchestrator Module. If the controller wants to modify that prevention rule somehow, it can change it by sending appropriate messages via the Orchestrator Module. Secondly, since this system provides a distributed detection/mitigation technique instead of a centralized one, the detection nodes should be widely distributed across the network. This is why we propose implementing the MiddleModule functions in edge-switches to maximize this distribution. Therefore, to obtain the best performance, and to fully utilize the system, the MiddleModule functions should be implemented in edge-switches that are directly connected to hosts. In this thesis; we assume that direct connections exist between edge-switches and all hosts, as shown in ***Figure 3.2***; and we suggest implementing the MiddleModule in this way to obtain the best performance.

3.2 SYSTEM REQUIREMENTS

Before explaining the system architecture and the framework in detail; we note that there are several critical points for a data plane-based Network/Transport-Level DDoS defense system for SDN architecture, which should be considered carefully while designing such a system. These points are selected by considering the requirements of SDN architecture, properties of data plane devices, characteristics of DDoS attacks and basic network security approaches. To the best of our knowledge, these points have not been gathered together and stated clearly for data plane-based DDoS defense systems earlier in the literature. In this thesis, we highlight these points and we suggest that any data plane-based DDoS defense system should be designed by considering the factors listed below:

1. **Complexity of Algorithms:** While designing a data plane-based defense system, the detection and prevention algorithms should be made carefully. Although complex algorithms provide high defense strength, or they may cover a large set of attacks; such algorithms may cause severe load for data plane devices. Firstly, data plane devices have limited resources; hence CPU or memory would be the bottleneck

if complex algorithms are used. Secondly, incoming traffic rate to a data plane device is still considerably high, hence the required processing time to analyze incoming traffic could become high if complex algorithms are implemented in data plane devices. This may cause high packet forwarding delays or even packet drops, therefore, one should choose low complexity detection and prevention algorithms.

2. **Implementation Style of the Defense System:** A data plane-based DDoS defense system for SDN can be implemented in the switch in a serially or parallel connected fashion. When it is implemented in serial mode, a malicious packet is dropped as soon as it is labeled as malicious before forwarding it, which slightly increases the detection rate of the defense system. However, since all packets have to be processed before forwarding, packet forwarding delay will increase considerably. When the defense system is implemented in parallel to the forwarding circuit; even if the received packet is labeled as malicious, instead of this specific packet, the next received packet can be dropped because the packet labeled as malicious has already been forwarded. Therefore, although few malicious packets are missed by the defense system decreasing the detection rate a little bit, there will be nearly no additional packet forwarding delay.

3. **Multiple-Location Defensive Approach:** Multiple-Location Defensive approach means placing or operating different defense algorithms to different locations in the network. For a data plane-based DDoS defense system, detecting all Network/Transport-Level DDoS attack types could be a challenging problem. The reason is that there are several types of attacks and some attacks can be detected near the source side, while others can be detected accurately only within the network or at the destination side. For example, at the source side, a protocol exploitation attack can easily be detected while a reflection attack cannot be detected at all. A data plane device, by itself, does not have a global view of the network and it is not aware of the traffic passing through the other switches. Therefore, to detect all Network/Transport-Level DDoS attack types, either this information should be sent to other switches, which increases the communication overhead of the system dramatically and increases the complexity of the system, or this problem should be solved by a careful implementation of multiple-location defensive approach. While

designing a data plane-based DDoS detection/mitigation system, the need for multiple-location defensive approach should be considered.

4. **Allowable data rate:** Today, the traffic generated by a single host is increasing due to advanced networking technologies. In addition, in Network/Transport-Level DDoS attacks, one of the main purposes of a malicious host is to generate as many packets as possible. Modern DDoS defense mechanisms should be capable of handle much higher data rates when compared with earlier methods. Hence, the maximum allowable malicious traffic rate should be taken into consideration while designing a defense system and such defense systems should be tested and evaluated using high data rates.

5. **Communicating with the controller:** In SDN, the intelligent part of the network is the controller in terms of forwarding. Any action regarding the forwarding operation should be handled in coordination with the controller. Data plane-based DDoS detection and prevention methods suggest that a data plane device detects the malicious activity and mitigates it by itself. However, the data plane devices should still inform the controller device about the detection and prevention decisions. Data plane devices should also be able to receive mitigation rules from the controller such as white or black lists. Furthermore, in SDN, a network operator is able to control and configure the entire network via the controller; therefore, a data plane-based defense system should also be configurable over the controller.

6. **Targeted DDoS Attack Types:** Selecting a correct target attack set is an important issue for network security systems. Targeting a large set of different attack types is not a good idea especially for data plane-based defense mechanisms because this could increase the complexity of the system dramatically or decrease the effectiveness. On the other hand, targeting only a small subset of an attack type is also not a good idea because network designers use a security system generally to avoid a specific attack type. If the used defense system does not cover all attacks in that specific attack type, the network designer needs to implement another defense system. Using similar defense systems, working on the same devices, controlling the same traffic and checking similar signatures, would generally result in poor performance. In addition, such usage of data plane-based defense systems exhausts the resources of the data plane devices even more and may add high amount of delay

to forwarding operations. Therefore, a defense system should clearly define its targets and while choosing its target, it should consider trade-off between the completeness of the solution and the complexity of the system.

7. **Detecting Source and Victim of Malicious Activity:** For effective mitigation, a defense system should provide information about the source and the victim of a malicious activity when it detects a DDoS attack. The type of the DDoS attack should also be detected. This information is valuable to mitigate the attack accurately. If a defense system detects an attack but does not provide such information, then the attack traffic cannot be prevented accurately.

In MiddleModule system, the detection functionality is provided by using lightweight algorithms to minimize the complexity of the system and to maximize the malicious traffic rate that can be detected. The packet processing circuitry is implemented in parallel to the forwarding circuitry to minimize forwarding delay. When a packet arrives at an edge-switch, it is mirrored to the detection block; while the original packet is forwarded towards the destination. To provide high defense strength against all types of Network/Transport-Level DDoS attacks, we use multiple-location defensive approach in our MiddleModule system. Reflection attacks are detected at the destination side while other attack types are detected at the source side of the messages. In this system, edge-switches apply detection functions to the incoming packets and decide mitigation rules. If desired, edge-switches can inform the controller about these mitigation decisions with periodic or sporadic messages. In addition, the controller can modify these decisions by adding or removing some mitigation rules. Finally, when the MiddleModule system detects a DDoS attack, it provides the DDoS attack type and the source and the victim of that malicious activity as an output.

The MiddleModule system is designed to defend SDN architecture against Network/Transport-Level DDoS attacks. As was mentioned earlier, DDoS attacks are classified into two categories, Network/Transport-Level DDoS Attacks and Application-Level DDoS Attacks. The former can have more devastating impacts on SDN when compared with the latter. Unlike Application-Level DDoS Attacks, the former is noticeable at network level and they can be detected with considerably

simpler detection algorithms, more suitable for data plane devices. Therefore, the MiddleModule system targets only Network/Transport-Level DDoS attacks. As was explained earlier, all Network/Transport-Level DDoS attack types can be very problematic for an SDN network, and all these attack types have similar characteristics. Hence, our MiddleModule system provides a detection/mitigation capability against all Network/Transport-Level attack types. Application-Level DDoS attacks are not within the scope of the MiddleModule system.

3.3 SYSTEM FRAMEWORK

The MiddleModule framework suggests implementing several functional blocks in the data plane for monitoring, detection and prevention purposes; and also, one block in the control plane for orchestration and configuration purposes.

3.3.1 MiddleModule Blocks in Data Plane

The MiddleModule framework suggests adding several blocks to edge-switches, to provide monitoring, detection and mitigation capabilities; which are namely ‘Packet Mirror and Attack Prevention Block’ and ‘Packet Processing Block’. For cooperation of the edge-switches with the controller in detection and prevention activities, ‘Control Block’ is also added to edge-switches. Therefore, three functional blocks in total are added to each edge-switch. ***Figure 3.3*** shows the placement of these blocks and their connectivity.

3.3.1.1 Packet Mirror and Attack Prevention Block

Packet Mirror and Attack Prevention Block is responsible for two basic operations, namely, mirroring the incoming packets to the Packet Processing Block and dropping the incoming packet if its source address or its ingress port is in the *Malicious List*. When a packet enters an edge-switch, it directly goes into this block, and firstly the *Attack Prevention* sub-block controls the packet. If the source address of this packet is in the *Malicious List*, then this packet is dropped without sending the packet to any other blocks. If it is not in the list, then the *Packet Mirror* sub-block receives the

packet. In this sub-block, the header portion of the packet is mirrored through the Packet Processing Block while the entire original packet is directed to the forwarding circuitry. To give further details, the sub-blocks will be explained separately.

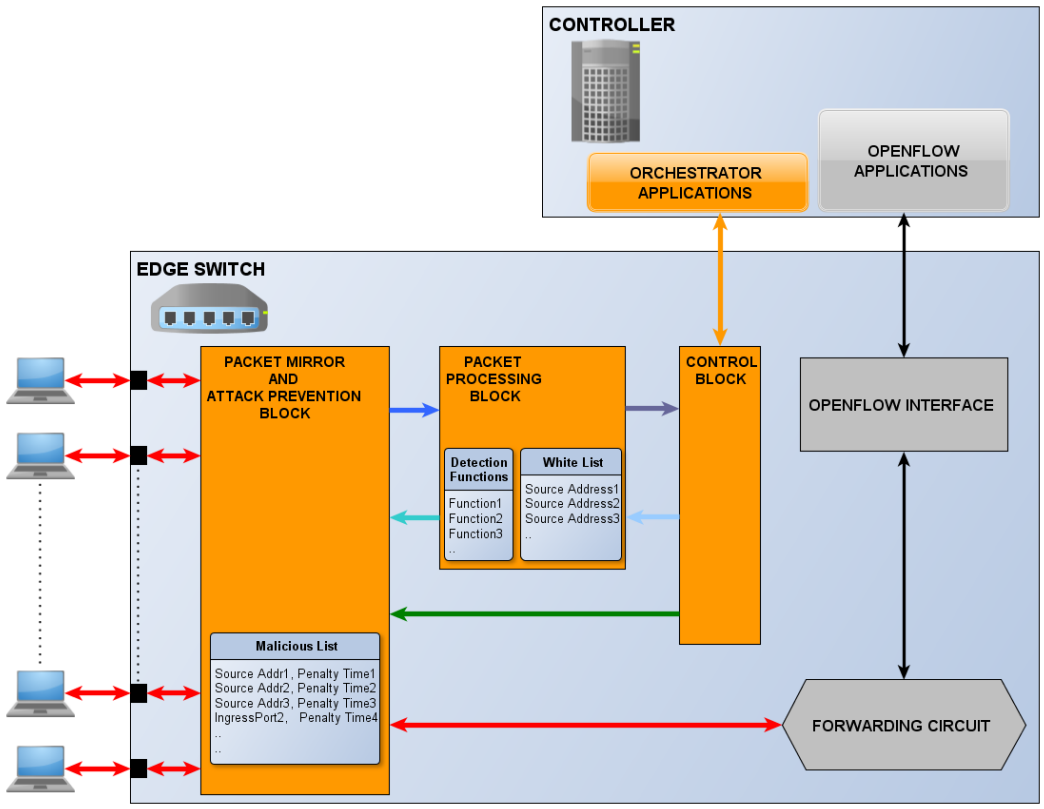


Figure 3.3: MiddleModule Blocks

Attack Prevention sub-block is responsible for performing the appropriate mitigation action according to the malicious detection results. It is the first block of the MiddleModule system that receives an incoming packet, as shown in ***Figure 3.3***. The reason is that if the received packet is coming from an already recorded malicious user, then there is no need to re-analyze this packet. This sub-block has a *Malicious List*, which includes the source addresses or ingress ports of the previously recorded malicious users. When Packet Processing Block detects any malicious activity, it sends the source address or the ingress port of that activity to the Attack Prevention sub-block, and this sub-block stores it in the *Malicious List*. In addition, if the controller sends any malicious source address to an edge-switch, Control Block

receives this information and sends it to the *Attack Prevention* sub-block. This sub-block stores the malicious address in its *Malicious List*.

Packet Mirror sub-block is located near the ingress ports of an edge-switch as shown in ***Figure 3.3***. When a packet arrives to an edge-switch from any ingress port, either from a host or a core switch, and if the *Attack Prevention* sub-block does not drop the packet, then *Packet Mirror* sub-block receives the packet. In this sub-block, the header portion of the packet is mirrored through the Packet Processing Block while the entire original packet is directed through the forwarding circuitry. The detection algorithms deployed in the Packet Processing Block can detect malicious activities by analyzing only the header portions of the packets. Mirroring only the header portions, instead of the entire packets, minimizes the communication overhead in the switch and minimizes the input queue utilization of the Packet Processing Block.

Packet Mirror sub-block is the packet sampling block of the overall MiddleModule system. There are different techniques for packet sampling; such as, sampling packets with a probability or sampling packets randomly or sampling all incoming packets. In these techniques, as the sampling rate increases, the number of analyzed packets increases, and eventually, the probability of missing any malicious packet decreases. In the MiddleModule system, *Packet Mirror* sub-block samples all the incoming packets, to maximize the malicious detection rate.

The Packet Mirror and Attack Prevention Block causes the only delay of MiddleModule system on the packet forwarding path and it is very limited; in fact, it is ignorable, as will be shown in Chapter 5. The required resources for this block are also very small. The analysis of required resources, scalability analysis and implementation suggestions can also be found in the evaluation part in Chapter 5.

3.3.1.2 Packet Processing Block

Packet Processing Block provides the detection capability to the edge-switches; hence, it could be considered as the intelligence added to the edge-switches in terms of detection operations. This block receives packet headers from the *Packet Mirror*

sub-block and processes them by using several Network/Transport-Level DDoS detection algorithms. Therefore, Packet Processing Block includes several sub-modules, which are *Input Queue*, *Detection Engine* that includes processing and storage infrastructure and *Detection Algorithms*.

When the header portion of any packet is received by the Packet Processing Block, it is stored in the *Input Queue*. This is designed as a "first-in first-out" queue. Although other scheduling techniques can be used, this is preferred due to its ease of use.

In Packet Processing Block, flow-based features are recorded and are used in malicious detection algorithms. Flows are created based on source or destination addresses, depending on the *Detection Algorithm*. Different features are recorded for different detection algorithms; for example, in SYN Flood, the number of received SYN packets is recorded for each source address-based flows while in Reflection detection, the number of received packets with different source IP addresses is stored for each destination address based flow. These features are stored in the *Features Table*, located in the *Detection Engine*. When an entry, i.e., the header portion of a packet, is dequeued from the *Input Queue*, associated flow information is found and the corresponding features of the flow are received from the *Features Table*. These features are directed to the *Detection Algorithms* and by analyzing them; the appropriate detection decisions are taken. After that, the *Features Table* is updated accordingly. If any malicious activity is detected for that flow, the source address of the malicious activity is stored in the *Malicious List*.

Since detection operations are handled very close to hosts, the size of *Features Table* could be very small. There are several reasons for this. Firstly, a small number of feature types are enough to make an accurate detection since *Detection Algorithms* collect features directly from the host connections. Secondly, the length or the window size of the collected features could be small, and new records are overwritten on the existing ones. Therefore, only very recent features are analyzed by detection algorithms. Small window size is enough for accurate detection because of the characteristics of the Network/Transport-Level DDoS attacks. When any host generates such attack, there will be definite changes in the recent packet features,

which will be then be detected much easily. However, obviously, the window size should still not be selected too small in order to avoid false positives.

In Packet Processing Block, it is critical to differentiate malicious from anomalous to minimize the false positive rates. In MiddleModule system, in addition to the architectural precautions, an additional false positive avoiding mechanism is used. In networking, simple anomalous activities do not tend to repeat themselves while the Network/Transport-Level DDoS attacks tend to last long, and they repeat themselves. In MiddleModule system, when a malicious activity is detected for the first time, the source address of that malicious activity is not stored directly in the *Malicious List*; instead, it is stored in a *blacklist table*, located in the *Detection Engine*, for a *penaltyTime*. If *Detection Algorithms* detect the same malicious activities originating from a specific source within the *penaltyTime*, then this source address is removed from the *blacklist table* to the *Malicious List* for another penalty time. If any packet is received from this source address during the associated penalty time, the packet is dropped. Further analysis of the Packet Processing Block regarding the scalability, performance analysis and the detection/prevention delays can be found in the evaluation part in Chapter 5.

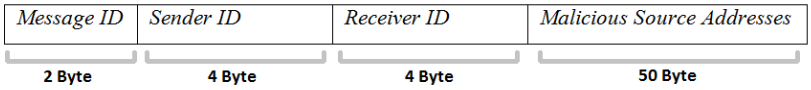
3.3.1.3 Control Block

Control block is used for communication between the MiddleModule blocks running on edge-switches and the MiddleModule blocks running on the controller. This block is critical for the MiddleModule system, because the system is designed for SDN and in SDN; controller decides all forwarding/routing related decisions. As was mentioned earlier, when an edge-switch detects any malicious activity and starts dropping the packets, the controller should be aware of this activity. The controller should also be able to define a whitelist or a blacklist to any edge-switch. Moreover, the controller should be able to enable or disable the entire MiddleModule operation of any edge-switch in the network or configure the detection algorithm variables.

Control Block is responsible for sending appropriate messages to the controller and for receiving and processing messages coming from the controller. Control Block has

a second purpose. When a reflection attack is detected by an edge-switch, the Control Block of that edge-switch is used for broadcasting the detected malicious source addresses to other switches. For this, the edge-switch sends a message to the controller via this block and the controller distributes this message to other edge-switches. Hence, edge-switches located very close to the malicious host start dropping messages received from these hosts and the malicious traffic is dropped without entering into the network.

To provide such connectivity, we define a communication protocol between the edge-switches and the controller, called *MiddleModule Message Protocol*, as shown in ***Figure 3.4***. The messages are exchanged over a dedicated communication channel between the Control Block and the controller as was seen in ***Figure 3.3***. For each message type, there is a predefined message ID. As an example, to disable the detection/mitigation blocks of a switch the controller sends a message with ID 0x01. This block can also be used for generating and handling *port detection messages*. *Port detection messages* are used for edge-switch port classification, which will be explained in Chapter 4. Any desired command or information can be transferred to the receiver with appropriate message ID (see ***Figure 3.4***). The MiddleModule messages are TCP messages with data payload size of 60 bytes. This protocol can be modified according to further needs. The overhead of this messaging can be ignored. Further analyses of MiddleModule message communication overhead is provided at the evaluation part in Chapter 5.



Message ID	Sender ID	Receiver ID
0x01 => disable MMFunctions 0x02 => inform the controller about the Malicious Source Address 0x03 => distribute the Malicious Source Address 0x04 => port detection	*Unique number for each device*	*Unique number for each device*

Figure 3.4: MiddleModule Message Protocol

3.3.2 MiddleModule Blocks in Control Plane

3.3.2.1 Orchestrator Block

Orchestrator Block is used to communicate with the Control Blocks of the edge-switches; communication taking place as illustrated in ***Figure 3.4.*** The communication behavior of this block is similar to the one explained in Control Block section. Orchestrator Block is designed as an application that runs on top of the controller. This block provides an ability to the network operator to configure or control the MiddleModule defense system running on edge-switches. In addition, network operator can observe the detection decisions made by the MiddleModule system. The network operator can reassign threshold values to detection algorithms to tune the performance of the system or to change the system's behavior, can change penalty time, can add idle time, etc. If the controller gets overloaded somehow, the Orchestrator Block can be terminated; which would not affect the overall performance of the MiddleModule system dramatically.

3.3.3 MiddleModule Operation

When a packet arrives to an edge-switch, it is transferred directly to the Packet Mirror and Attack Prevention Block, as shown in ***Figure 3.5***. *Attack Prevention* sub-block receives the packet and checks if the source address is in the *Malicious List* or not. If yes, the packet is dropped without any further analysis. This will protect the switch resources, controller resources, and also the victim host from malicious activities. In addition, no MiddleModule malicious detection resource is utilized for further analysis of this packet. If the source host of the packet is not in the *Malicious List*, then the *Packet Mirror* sub-block receives the packet and mirrors the header portion of the packet through the Packet Processing Block and sends the entire packet through the Forwarding Circuit. If pipelining is used in the system, since the Packet Mirror and Attack Prevention Block becomes idle, the system can receive and process a new packet from the network. Forwarding Circuit operates as in a standard OpenFlow switch since MiddleModule system is making any modification in the OpenFlow and Forwarding components of the switch. When the header portion of

the packet arrives at Packet Processing Block, it is stored in the *Input Queue*. Whenever the *Detection Engine* in the Packet Processing Block becomes idle, the first entry in the *Input Queue* is dequeued and the *Detection Engine* receives the entry, namely the header portion. First, *Detection Engine* finds the corresponding flow for the entry, then it checks the *Features Table* and retrieves the corresponding features of the received header. These features include several lists and values based on previously history of received packets for the flow which are used in the *Detection Algorithms*. Detailed explanations regarding the recorded features can be found in the *Detection Algorithms* part of the thesis, in Chapter 4. *Detection Engine* runs several lightweight *Detection Algorithms* and if any one of them detects malicious activity, it stores the source address of the packet to the *blacklist table* for a *penaltyTime*. If the address of the source host is already in the *blacklist table*, then *Detection Engine* sends this address to the *Attack Prevention* sub-block to store it in the *Malicious List* for another penalty time and removes this address from the *blacklist table*. The F*eatures Table* in the Detection Engine is updated with the recent features. If a malicious activity is detected, the detection decision is also forwarded to the Control Block. This block can then send the source address of this malicious activity to the Controller and the Controller can distribute it to all edge-switches.

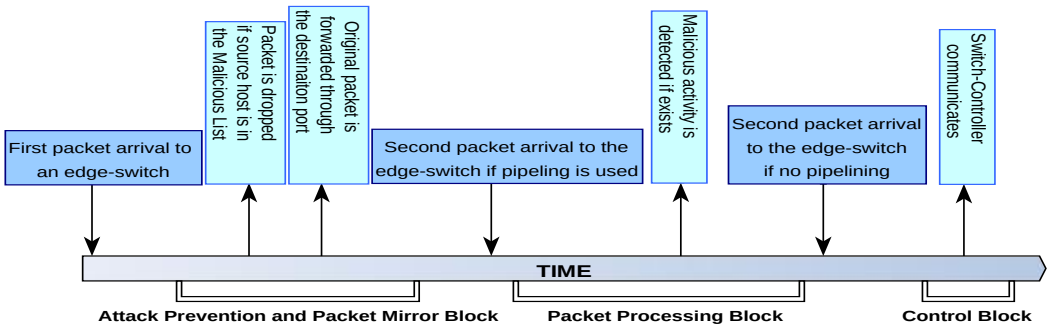


Figure 3.5: MiddleModule Operation Diagram

CHAPTER 4

DETECTION ALGORITHMS

In the MiddleModule system, to detect the Network/Transport-Level DDoS attacks, several detection algorithms are designed and implemented within Packet Processing Blocks of the edge-switches. As was explained in the background part in Chapter 2, there are mainly five types of Network/Transport-Level DDoS Attacks. In this thesis, we provide detection algorithms against the well-known samples of each Network/Transport-Level DDoS attack types, as follows:

1. **IP Spoofing Attack Type:** *Source IP Spoofing Detection algorithm*
2. **Protocol Exploitation Attack Type:** *SYN Flood Detection algorithm*
3. **Flooding Attack Type:** *ICMP Flood and UDP Flood Detection algorithm*
4. **Reflection Attack Type:** *ICMP Reflection Detection algorithm*
5. **Amplification Attack Type:** *Broadcast Amplification Detection algorithm*

In this chapter, a detailed explanation of the above detection algorithms is given.

4.1 General Detection Algorithm Approach in MiddleModule System

Network/Transport-Level DDoS attacks have two critical features, which distinguish them from other network incidents. Firstly, these attacks cause large network traffic; hence, they increase the traffic volume significantly. Secondly, most of the packets in the network become very similar to each other regarding some packet features; hence, variation of some packet properties decreases or increases noticeably. In an organized and massive DDoS attack, these changes can be recognized along the entire network, while in a smaller attack; they are detected at least on the communication channel of the responsible malicious host. If an attack is so small that it does not cause any such alteration in the traffic, then this attack does not pose any

threat towards the SDN network; hence, it is not within the scope of this study. For example, during a DoS SYN Flood attack, the SYN packet volume and the ratio of the number of SYN packets to the number of all packets measured at the communication channel of the malicious host increase noticeably. On the other hand, during an organized DDoS ICMP Reflection attack made with reflectors, the volume of the ICMP packets in the network increases while the variations of the destination addresses, packet byte lengths and packet protocols of the packets in the network decrease. Therefore, in Network/Transport-Level DDoS detection algorithms, to detect if there is a DDoS attack on the network and if so, to detect the source and the victim host and the attack type, we can seek the changes in some features, as shown in ***Figure 4.1***.

In the MiddleModule system, the detection algorithms are designed with this approach in mind. The detection algorithms provided in our system use simple feature extraction methods and basic threshold-based comparison techniques (see ***Figure 4.1***) hence are lightweight ones. Even if they are lightweight; since they are implemented in data plane devices, and they are seeking very specific changes in the characteristic features for DDoS attacks and since they analyze each packet received from connected hosts; they provide remarkable detection performances as will be shown in the evaluation part in Chapter 5. In addition, using multiple-location defensive approach is critical in obtain high detection performance using such lightweight detection algorithms.

For such lightweight techniques, the false positives could of course be a major problem. However, to minimize false positive rates, some further capabilities are inserted in the MiddleModule system. The threshold values used are selected by considering both the anomalous and normal traffic characteristics. In fact, to be in the safe side in terms of false positives, the implemented threshold values are chosen with a margin on the calculated threshold values. As our evaluation results suggested in Chapter 5, the MiddleModule system easily detects the DDoS attacks with negligible false positive rates.

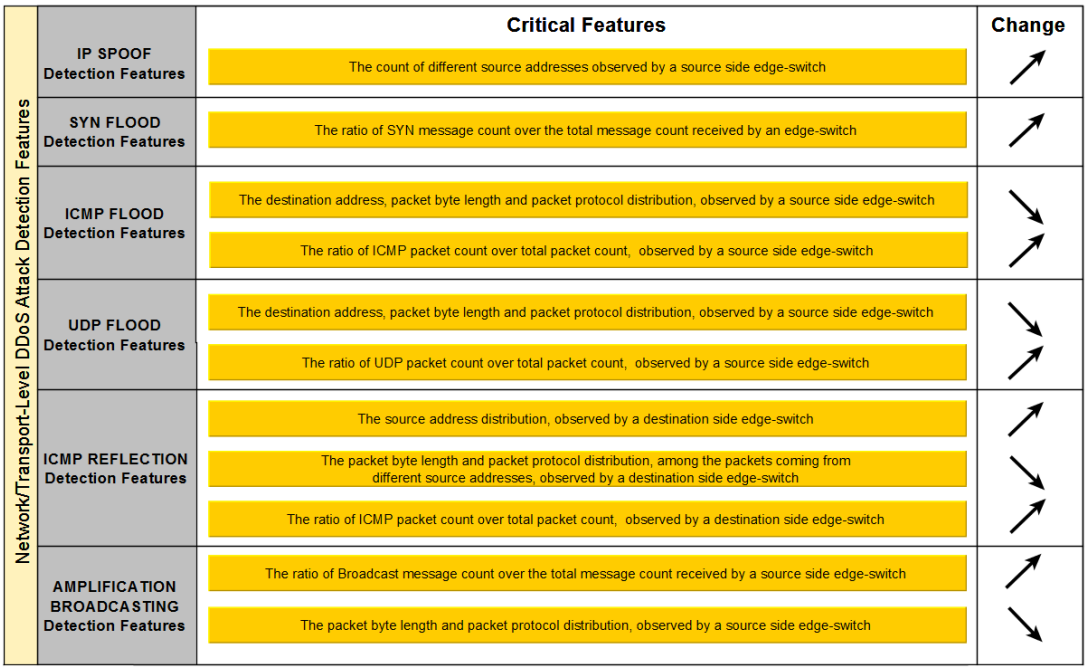


Figure 4.1: Network/Transport-Level DDoS Attacks Detection Features

In MiddleModule system, there are several different attack detection algorithms to detect different Network/Transport-Level DDoS attack types. These algorithms are classified into two different sets *source-side algorithms set* and *destination-side algorithms set*. Although both of these sets are implemented at all edge-switches, only one set is applied to a received packet, according to the incoming port of the packet to the edge-switch. The ingress ports of a switch are categorized as *host ports* and *network ports*. If an ingress port of an edge-switch is directly connected to a host, then this port is called *host port*. On the other hand, if an ingress port of an edge-switch is connected to another switch, then this port is called *network port*. When a packet is received from a *host port*, then the edge-switch applies the *source-side algorithms set;* or if a packet is received from a *network port*, then the edge-switch applies the *destination-side algorithms set*. Reflection Attack Detection Algorithms constitutes the *destination-side algorithms set*, while the other attack detection algorithms constitute the *source-side algorithms set*. In the MiddleModule system, we assume that if a host is a malicious host, most of the traffic generated by this host belong to that malicious activity, which is a valid assumption when the characteristics of the Network/Transport-Level DDoS attacks are considered. To the

best of our knowledge, this assumption is utilized in most of the similar works found in the literature [4], [39].

To detect *host ports* and *network ports* of an edge-switch, the MiddleModule system uses *port detection message*, defined in *MiddleModule Message Protocol*. During the initialization of the network, each edge-switch generates *port detection message*, which is forwarded to other edge-switches over the core switches. Finally, all edge-switches receive such *port detection* message over their *network ports* and the remaining ports of that edge-switch are labeled as *host ports.* We prefer this because the MiddleModule system is implemented in our work on relatively small networks. If the network is a large one, to minimize communication overhead, different approaches can be used; for example, this information can be provided by the controller to edge-switches by using the global network wide view.

4.2 Protocol Exploitation Attack Detection Algorithm

In this attack type, a malicious host generates packets to abuse some properties of the communication protocols to exhaust the resources of the victim host. In this thesis, SYN Flood attack is considered as a sample of protocol exploitation attack type and a detection algorithm is provided against SYN Flood attacks, as explained in ***Algorithm 4-1***. To detect SYN Flood attack, at the source side of packets, the ratio of SYN packet count over total packet count is calculated for each host having different source IP addresses. Hence, this algorithm analyzes the variation of the message type.

For normal TCP traffic, the ratio of the count of SYN messages and the total message count should be smaller than a threshold. If the threshold is chosen carefully to cover network anomalies, then when this ratio exceeds the threshold, it can be deduced that there is a SYN Flooding activity from a host through a victim server.

In SYN Flood detection algorithm, when a packet, *p,* arrives at an edge-switch, the incoming port of the packet, *i,* is checked to see if the packet is coming from a *host port* or a *network port*. If it is coming from a *host port*, then the packet, *p*, is stored in

a list, $pcktWindow_s$ with the index number of $lastEntry_s$. For different source IP addresses, different lists are used. If the list is full, then the new packet is overwritten the oldest packet; hence, the list always includes most recent packets, which is critical for detection performance. After storing *p*, the total number of packets in $pcktWindow_s$ list, t_s, and the total number of SYN packets in the $pcktWindow_s$ list, s_s, are calculated. Then several comparison operations are performed for malicious activity detection. Firstly, the ratio of the total number of packets in the $pcktWindow_s$ list and *windowSize* is calculated and it is compared with a *saturationThreshold* to avoid immature decisions. If there are sufficient number of packets, the ratio of the number of SYN packets, s_s, and the total number of packets in the $pcktWindow_s$ list, t_s, is compared with the *synFloodThreshold* value to check if an enormous increase in the number of SYN packets exists or not. If so, the responsible source IP address is added into a *blacklist table* in the edge-switch for a penalty time and all statistics collected so far for the associated host is cleared.

The assignment of *synFloodThreshold* value is an important issue for detection performance of the algorithm. To assign a correct value, both standard traffic scenarios and anomaly scenarios should be considered. In a standard TCP connection, including 3-way handshaking, data transaction and FIN-ACK parts, the ratio of the number of SYN packets to the number of total packets is less than 1/5. In fact, this ratio gets even smaller if there are different message protocols on the network other than TCP. In some cases, there could be network anomalies, and this may increase the number of SYN packets over the total number of packet. However, even during these anomalies, this ratio should not be larger than a threshold for a long time; such as 1/2. Therefore, an optimal *synFloodThreshold* value can be found according to the network topologies, by considering the expected normal traffic and possible network anomalies.

Algorithm 4-1: SYN Flood Detection Algorithm
1 arrival of new packet *p*
2 *s* ← the source IP address of *p*
3 *i* ← the ingress switch port of packet *p*
4 if *i* is a host port **then**
5 $pcktWindow_s[lastEntry_s] \leftarrow p$

6	**if** $lastEntry_s < windowSize$ **then**
7	increment $lastEntry_s$
8	**else**
9	$lastEntry_s \leftarrow 0$
10	$t_s \leftarrow$ total number of packets in the $pcktWindow_s$ set
11	$s_s \leftarrow$ total number of SYN packets in the $pcktWindow_s$ set
12	**if** $(s_s / t_s > synFloodThreshold) \wedge (t_s / windowSize > saturationThreshold)$ **then**
13	p is malicious
14	$t_s \leftarrow 0$
15	$s_s \leftarrow 0$

4.3 IP Spoofing Attack Detection Algorithm

In this attack type, a malicious host generates packets with forged source IP addresses. Generally, source IP spoofing is not classified as a separate attack type but in this thesis we consider source IP spoofing as a separate Network/Transport-Level Attack type and a detection algorithm is provided against IP Spoofing attacks as shown in ***Algorithm 4-2***.

To detect this type of attack at the source side of packets, the source IP addresses of the incoming traffic are monitored to detect any significant increase in the variation of the source IP addresses of incoming packets. If the number of packets, with different source IP addresses, coming from a single switch *host port* increases above a threshold value, then the host, connected to this *host port* is added to the IP Spoof attackers black list.

In IP Spoofing attack detection algorithm, when a packet, *p*, arrives to an edge-switch, the incoming port, *i*, is controlled if it is a *host port* and if so, the packet, *p*, is stored in the $pcktWindow_i$ list with the index number of $lastEntry_i$. In this algorithm, for each different switch ingress port, different statistics and different lists are stored; hence, ingress port-based flows are generated. Then, the ratio of the total number of packets in the $pcktWindow_i$ list and the *windowSize* is compared with the *saturationThreshold* to avoid immature decisions. In addition, source IP addresses of packets in the $pcktWindow_i$ list are analyzed; and the number of different source IP addresses, stored in the $pcktWindow_i$ list, is compared with the *spoofThreshold*. If it is bigger than the *spoofThreshold*, this means that there are too many packets with

the different source IP addresses arriving from a single ingress port of the edge-switch. This algorithm is designed assuming that the edge-switches are very close to the hosts and only one host is connected to one ingress port of the switch. Hence, if both comparisons show that the values are larger than the thresholds, it is deduced that the host, connected to that ingress port of the edge-switch, sends IP Spoofing traffic into the network. The edge-switch stores the ingress port into the *blacklist Table* for a penalty time, clears all the statistics kept for that port, and restart monitoring it. If IP spoofing activity is detected at the same ingress port in that penalty time, then this port is added to the *Malicious List* and the packets coming from that port is dropped for a penalty time.

Algorithm 4-2: IP Spoof Detection Algorithm

1 arrival of new packet p
2 $s \leftarrow$ the source IP address of p
3 $i \leftarrow$ the ingress switch port of packet p
4 if i is a host port **then**
5 $pcktWindow_i[lastEntry_i] \leftarrow p$
6 **if** $lastEntry_i < windowSize$ **then**
7 increment $lastEntry_i$
8 **else**
9 $lastEntry_i \leftarrow 0$
10 $t_i \leftarrow$ total number of packets in the $pcktWindow_i$ set
11 $diffSrc_i \leftarrow$ different source IP addresses of packets in the $pcktWindow_i$ set
12 **if** (*size of* $diffSrc_i$) > *spoofThreshold*) ∧ (t_i / *windowSize* > *saturationThreshold*) **then**
13 p is malicious
14 $t_i \leftarrow 0$
15 clear($diffSrc_i$)

4.4 Flooding Attack Detection Algorithm

In Flooding attack type, a malicious host generates a large number of redundant packets towards a victim host to exhaust its resources and communication bandwidth. In this thesis, ICMP Flooding and UDP Flooding are considered as samples of flooding attack type and a detection algorithm is provided against ICMP Flooding and UDP Flooding attacks, as explained in ***Algorithm 4-3***.

To detect Flooding attack, the number of packets sent from a single source host to a destination is checked. The properties of the packets received from that source host are also analyzed. During a Flooding attack, a source host sends a large number of packets towards a destination and the variation of the destination IP addresses of the packets received from that source host decreases noticeably. Furthermore, during such attacks, the packet byte length and the packet protocol of most of the packets received from that source host become identical. Although other similarities could be observed during such attacks, other than packet byte length and packet protocol, other similarities are not considered in MiddleModule system for convenience. The packet byte lengths and packet protocols are named as packet properties in this algorithm.

In Flooding Detection Algorithm, when a packet, *p*, arrives at an edge-switch from its *host port*, it is stored in the $pcktWindow_s$ list with the index number of $lastEntry_s$. Different statistics and lists are used for different source hosts. After storing the packet in the list, the $lastEntry_s$ value is incremented if it is smaller than the *windowSize* value. Then, the ratio of the number of packets in the $pcktWindow_s$ list and the *windowSize* is calculated and compared with a *saturationThreshold*. If there are enough number of packets in the list, then the number of list packets having the same destination address *p* is found as $occDest_d$. In addition, the number of list packets having the same properties as *p* is also found, $occProp_{props}$. After that, the ratio of $occDest_d$ and the total number of packets in the $pcktWindow_s$ list is calculated. And this ratio is compared with the *floodThreshold* value to check the distribution of destination IP addresses of the packets coming from that source host. After that, the distribution of the properties of the packets received from that source host is calculated by finding the ratio of $occProp_{props}$ value to total number of packets in the $pcktWindow_s$ list. This ratio is compared with the *floodThreshold* value. If the comparison results imply that there is flooding attack from that source host, then source IP address of the host is stored in the *blacklist Table* and the monitored features are cleared to their default values and monitoring operation restarts.

There is a critical detail in the Flooding attack detection algorithm. If the victim host sends reply messages to the incoming Flooding packets, it could be labeled as

malicious, too. To avoid this problem, when a packet is received from a host port for the first time, the detection algorithm checks if this is the initial message or a response message. To do so, when a packet is received from *host* port, it is searched that if the destination address of the packet is in the different destination address list, building the $occDest_d$ value, or not. If it is not in the list, then the $diffSrc_s$ list, from the reflection attack detection algorithm, is checked to see if the destination address of that packet is in the list or not. If it is, then it is deduced that the packet is sent as a reply to a message; hence flooding detection algorithm is not applied to the packet.

Algorithm 4-3: Flooding Detection Algorithm

1 arrival of new packet *p*
2 *s* ← the source IP address of *p*
3 *d* ← the destination IP address of *p*
4 *i* ← the ingress switch port of packet *p*
5 *prop* ← the property of *p*
6 $diffDestAddrList_s$ ← the list of different destination IP addresses
7 **if** *i* is a host port **then**
8 **if** (*d* is in $diffDestAddrList_s$) ∧ (*d* is in $diffSrc_s$) **then**
9 **break**
10 $pcktWindow_s[lastEntry_s]$ ← *p*
11 **if** $lastEntry_s$ < *windowSize* **then**
12 increment $lastEntry_s$
13 **else**
14 $lastEntry_s$ ← 0
15 t_s ← total number of packets in the $pcktWindow_s$ set
16 $occProp_{props}$ ← the number of packets with property *prop* in the $pcktWindow_s$ set
17 $occDest_d$ ← the number of packets with destination *d* in the $pcktWindow_s$ set
18 **if** t_s / *windowSize* > *saturationThreshold* **then**
19 **if** ($occDest_d$ / t_s > *floodThreshold*) ∧ ($occProp_{props}$ / t_s > *floodThreshold*) **then**
20 *p* is malicious
21 t_s ← 0
22 $occDest_d$ ← 0
23 $occProp_{props}$ ← 0

4.5 Amplification Attack Detection Algorithm

In this attack type, a malicious host generates malicious traffic and it modifies some properties of the packets in that traffic to amplify the effects of the malicious activity. By using this attack, the attacker utilizes more network bandwidth of the victim host

than its own bandwidth to create the attack. In this thesis, Broadcast amplification attacks are considered as the sample of amplification type of attack. A malicious user broadcasts malicious messages through the network instead of directing these messages to a victim server, or a malicious user sends reflection packets to reflectors and makes them broadcast their replies. The main purpose of this attack is to amplify the effects of the attack for the underlying network and for the legitimate users. If a request message is broadcasted through the network, such as ICMP request and if the legitimate users respond to it, the effect of the attack becomes devastating. In MiddleModule system, to avoid broadcast amplification attacks, an algorithm is provided, which checks both the number of broadcast messages and the variation of the packet properties, as explained in ***Algorithm 4-4***.

This detection algorithm works as follows: when a packet arrives to an edge-switch from its *host port*, it is stored in the $pcktWindow_s$ list with the index number of $lastEntry_s$. The ratio of the total number of the packets in the $pcktWindow_s$ list and *windowSize* is calculated and it is compared with a *saturationThreshold*. If there are enough packets in the $pcktWindow_s$ list to make an accurate decision, the number of broadcast messages in the $pcktWindow_s$ list is found. Packet properties, packet byte lengths and packet protocols in the $pcktWindow_s$ list are also checked, and the number of different packet properties is computed. Then, the number of broadcast messages and the number of different packet properties in the list are compared with the total number of packets in the list. If these comparisons imply that there is a Broadcast amplification attack, then the source IP address of the responsible source host is stored in the *blacklist Table* first, and if that source is detected as malicious again then it is stored in the *Malicious List*. In this algorithm, source-based flows are generated, and packet monitoring and malicious detection are handled for each source-based flow.

Algorithm 4-4: Broadcasting Detection Algorithm
1 arrival of new packet *p* **2** *s* ← the source IP address of *p* **3** *i* ← the ingress switch port of packet *p* **4** *prop* ← the property of *p* **5 if** *i* is a host port **then**

6 $pcktWindow_s[lastEntry_s] \leftarrow p$
7 **if** $lastEntry_s < windowSize$ **then**
8 increment $lastEntry_s$
9 **else**
10 $lastEntry_s \leftarrow 0$
11 $t_s \leftarrow$ total number of packets in the $pcktWindow_s$ set
12 $b_s \leftarrow$ total number of broadcast messages in the $pcktWindow_s$ set
13 $occProp_{props} \leftarrow$ the number of packets with property *prop* in the $pcktWindow_s$ set
14 **if** $t_s / windowSize > saturationThreshold$ **then**
15 **if** $(b_s / t_s > amplificationThreshold) \wedge (occProp_{props} / t_s > amplificationThreshold)$ **then**
16 p is malicious
17 $t_s \leftarrow 0$
18 $b_s \leftarrow 0$

4.6 Reflection Attack Detection Algorithm

In this attack type, a malicious host sends several request messages towards the reflectors, with modified source IP address. Instead of writing its own IP address, the malicious user writes the IP address of the victim host at the source IP address part of the messages; therefore, the replies from the reflectors are sent to the victim host. In this thesis, ICMP Reflection attack is considered as a sample of reflection attack type and a detection algorithm is provided against ICMP Reflection attack, as explained in ***Algorithm 4-5***.

This attack type cannot be detected at the source side of a packet because this is a collaborative attack and it is generated by several separate nodes. In MiddleModule system, this attack is detected at the destination side of the packets. During a reflection attack, the number of packets with different source addresses increases at the edge-switch near victim host, so that the source IP address variation at the edge-switch increases above a threshold. In addition, the reflection attack traffic is in fact the reply messages generated from the reflector nodes to an attack trigger message received from a malicious host. Therefore, the reflection attack packets, have identical packet byte lengths and packet protocols. Although there are several other similarities in those messages, in MiddleModule system, only these two properties are considered for simplicity, and they are called as packet properties. Therefore, during a reflection attack, at the edge-switch near the victim host the variation of the

packet properties, whose source IP addresses are the same, decreases below a threshold.

In reflection detection algorithm, when a packet, *p*, arrives at an edge-switch, the incoming port, *i*, is controlled if it is a *network port* and if so, the reflection attack detection algorithm is applied on the packet. In this algorithm, destination address-based flows are generated unlike the previous algorithms, where source address-based flows were earlier. After receiving the packet, *p*, from a *network port*, before storing the packet in the $pcktWindow_d$ list, the existing $pcktWindow_d$ list is checked and different source IP addresses of the packets in the $pcktWindow_d$ list are stored in the $diffSrc_d$ list. The properties of these packets, which are included in $diffSrc_d$ list, are stored in $diffSrcProp_d$ list. The packet *p* is then stored in the $pcktWindow_d$ list using the index $lastEntry_d$ and $lastEntry_d$ is incremented if it is smaller than *windowSize*. Then, the source IP address of the received packet, *s*, is compared with the IP addresses existing in the $diffSrc_d$ list and if it is not in that list, *s* is added to the $diffSrc_d$ list; and the property of the received packet is added to the $diffSrcProp_d$ list. The ratio of the total number of the packets in the $pcktWindow_d$ list to *windowSize* is compared with *saturationThreshold* value. If there exist enough packets in $diffSrcProp_d$ list, then the ratio of the size of the $diffSrc_d$ list to total number of packets in the $pcktWindow_d$ list is calculated and this ratio is compared with *refSrcThreshold*. If it is bigger than the threshold, it means that there are too many packets having different source IP addresses sent to the destination host. Finally, the packet properties in the $diffSrcProp_d$ list are analyzed, and the number of packets having the same properties with others is found; and the maximum number of packets having same properties in the $diffSrcProp_d$ list is assigned to the $maxNumbSameProp_d$ value. If the ratio of $maxNumbSameProp_d$ value to the size of $diffSrc_d$ list is larger than *refPropThreshold*, this means there are too many packets, whose source addresses are different, but whose properties are similar to each other. Therefore, all such packets satisfying these conditions are considered as malicious and the source IP addresses of these packets are sent to other edge-switches over the controller. Hence, packets coming from these hosts are mitigated at the source-side.

Algorithm 4-5: Reflection Detection Algorithm

1 arrival of new packet p
3 $i \leftarrow$ the ingress switch port of packet p
2 $d \leftarrow$ the destination IP address of p
4 $s \leftarrow$ the source IP address of p
4 $prop \leftarrow$ the property of p
5 **if** i is a destination port **then**
6 $diffSrc_d \leftarrow$ different source IP addresses of packets in the $pcktWindow_d$ set
7 $diffSrcProp_d \leftarrow$ the properties of packets that build $diffSrc_d$ set
8 $pcktWindow_d[lastEntry_d] \leftarrow p$
9 **if** $lastEntry_d < windowSize$ **then**
10 increment $lastEntry_d$
11 **else**
12 $lastEntry_d \leftarrow 0$
13 $t_d \leftarrow$ total number of packets in the $pcktWindow_d$ set
14 **if** s is not in $diffSrc_d$ **then**
15 add s into $diffSrc_d$
16 add $prop$ into $diffSrcProp_d$
17 $maxNumbSameProp_d \leftarrow$ maximum number of the same properties in $diffSrcProp_d$ set
18 $samePropSrc_d \leftarrow$ source IP addresses of packets that build $maxNumbSameProp_d$
19 **if** t_d / $windowSize > saturationThreshold$ **then**
20 **if** $size\ of\ diffSrc_d / t_d > refSrcThreshold$ **then**
21 **if** $maxNumbSameProp_d / size\ of\ diffSrc_d > refPropThreshold$ **then**
22 $samePropSrc_d$ list is malicious
23 send $samePropSrc_d$ list to controller
24 $t_d \leftarrow 0$
25 clear $diffSrc_d$ set
26 clear $diffSrcProp_d$ set
27 clear $samePropSrc_d$ set
28 $maxNumbSameProp_d \leftarrow 0$

For example, consider a victim host, which is under an ICMP reflection attack; and the edge-switch directly connected to the victim host has *windowSize* of 100. Several reflector nodes, say 90 reflectors having different source IP addresses send malicious packets to the host. When the edge-switch stores 100 packets in its $pcktWindow_d$ list, let's say 95 packets are received from 90 different reflectors and five packets are received from two different legitimate hosts. Assume that 95 malicious packets have similar packet properties with each other, 2 packets out of 5 normal ones have similar packet properties with each other and the remaining 3 have different packet properties. In this example, the $diffSrc_d$ list will include 92 different source addresses

and *diffSrcProp$_d$* list includes 92 entries, which are the properties of the most recent packets received from those 92 different sources. For this simple example, assume the *saturationThreshold* value is 0.75 and the *refPropThreshold* is 0.8. First, the ratio of the number of packets in the *pcktWindow$_d$* list, and the *windowSize* value, is to be calculated and 100/100 is found as the result. Then, this ratio of one is compared with the *saturationThreshold* value of 0.75. Since there are enough number of packets in the list, the ratio of the *diffSrc$_d$* list size to the number of packets in the *pcktWindow$_d$* list is now calculated as 0.92 and it is compared with the *refPropThreshold* value. *diffSrcProp$_d$* list is checked for identical entries; and it is found that 95 entries and 2 entries similar packet properties. Since the maximum is 95, it is assigned to *maxNumbSameProp$_d$*. Then the ratio of *maxNumbSameProp$_d$* value and the size of *diffSrc$_d$* list is calculated as 1.03 (95/92) and this ratio is also compared to *refPropThreshold* value. Since both comparisons imply that there is a Reflection attack, then 90 source IP addresses of the packets building the *maxNumbSameProp$_d$* value are labeled as malicious.

CHAPTER 5

EVALUATION

An extensive evaluation is performed to examine the behavior of the MiddleModule system and to analyze its performance. In this chapter, the performance measurement metrics are discussed, the evaluation scenarios are explained in detail, and obtained test results are presented. The test results are compared with important similar studies and the advantages and disadvantages of the MiddleModule system are stated clearly.

The MiddleModule system is evaluated in two different platforms, namely OMNET++ 4.2 and Mininet. The OMNET++ is a discrete event simulator platform and Mininet is a network emulator platform that uses process-based virtualization technique. During MiddleModule system design and the detailed functional analysis, OMNET++ 4.2 simulator is used because of its flexible and modular nature. For the performance analysis of the MiddleModule system and the proof-of-work, Mininet network emulator platform is used because of its realistic and reliable nature.

To evaluate the performance of the MiddleModule system; in this chapter, the defense strength and the system performance degradation are measured using different packet generation rates and using different test scenarios. By considering the measurement results obtained, scalability, implementation complexity and compromise-ability of the MiddleModule system are discussed.

Our evaluations are performed on a single computer, having properties stated in ***Table 5.1****Error! Reference source not found.***.**

Table 5.1: Hardware Specifications

Processor Model	Intel i7-5500U CPU
Number of Cores	2
Number of Threads	4
Processor Base / Max Frequency	2.40 GHz / 3.0 GHz
Installed Memory	12 GB
Instruction Set	64 Bit

5.1 Evaluation Tools

5.1.1 Evaluation Platforms

A network system can be evaluated in a simulator platform or in a hardware testbed or in an emulator platform. There are many differences between these platforms, and the most critical ones are reliability, realism, reproducibility, flexibility and cost [43]. In this thesis, we concentrated on both simulation and emulation. Simulators can provide reproducible and realistic analysis; and they provide flexible and easily manageable network topologies and low-cost evaluation systems. Custom network systems can be created easily and be evaluated with custom topologies and custom communication protocols by using simulators. Despite these advantages, simulators do not use the same codes running on real networks; and their models for traffic generation, communication protocols or hardware devices may raise accuracy concerns, which decreases their reliability [43]. Emulators provide both flexible and reliable evaluation platforms; hence, they stand between the simulators and the hardware testbeds in terms of these properties. Emulators run real codes regarding network applications, OS kernel, etc. with interactive network traffic by creating virtual hardware; therefore, they provide reliable and accurate evaluation results. On the other hand, like simulators, custom topologies can be created with emulators and they are still low cost analysis systems [43].

Therefore, during the design of a network defense system, network simulators can be used because of their flexibility and the sense of realism provided by the simulators.

In addition, as suggested in [44], using simulators could be beneficial for functional analysis of network defense systems because of their flexible and manageable nature. After designing a network defense system and performing basic functional analysis, the network emulators can be used for detailed performance analysis of the network defense systems, and for proof-of-work. By using emulators, detailed and reliable analysis results can be collected, and the performance of the proposed system can be compared with similar studies. During evaluation of MiddleModule system, we follow this approach and use both OMNET++ and Mininet.

In this thesis, OMNET++ 4.2 is operated on Windows 10 operating system [45]. *Inet 2.0* library [46] is used to provide basic networking capabilities and *ofomnet* library [47] is used to provide SDN and OpenFlow capabilities. *Ofomnet* library provides SDN architecture with OpenFlow 1.0 protocol. Our MiddleModule system framework and the associated detection algorithms are created in OMNET++ 4.2.

The Mininet [48] GUI/X11, on the other hand, is built on VirtualBox, that runs on Windows 10. The Mininet version used in these tests is 2.2.2; and OpenFlow 1.0 is used in these tests. In the test scenarios, the Open vSwitch 2.0.2 and OVS-Controller 2.0.2 are used as OpenFlow network nodes. Various test topologies are built and a detailed performance analysis of the MiddleModule system is performed on this platform.

5.1.2 Traffic Datasets

Traffic datasets are important to obtain reliable evaluation results for network defense systems. Once a network defense system is created, it should be evaluated with realistic test scenarios, and the network traffic is an important part of the scenario. During the evaluation of the MiddleModule system, different traffic datasets are used for different test platforms. In the simulation phase, involving OMNET++, the NSL-KDD dataset is used to create legitimate traffic and some of the malicious traffic [49]. It is an improved version of the standard KDD'99 dataset [50]. The standard KDD dataset is generated at the MIT Lincoln Labs. A local area network that simulates a real military LAN environment was created and operated as

if it was a true networking environment. In addition, some DoS attacks were implemented on the network, along with other attack types. This network traffic was monitored, and features of each connection was recorded and presented in the KDD dataset. The NSL-KDD improves the original KDD dataset by removing redundant packets and duplicate records, etc. Although this dataset is considerably old, and it consists of synthetic traffic, we used this dataset in our simulations because it is widely used in the literature and it is convenient to use it in simulation environments. On the other hand, during our emulation phase that utilized the Mininet platform, traffic is generated by using a real traffic dataset received from MAWI Working Group Traffic Archive [51], which is a popular dataset widely used in the literature [2] [39]. MAWILab has been working for a long time to collect real traffic at backbones. They have been sharing real traffic datasets since 2002 up until today. The traffic is labelled as *benign* or *malicious*. In this thesis, the traffic, collected on Jan 12, 2014 is used and the legitimate traffic is generated by using the packets, labeled as *benign*, during the emulation phase.

5.2 Performance Metrics

There are several performance measurement metrics that should be considered while designing and testing a DDoS defense mechanism. As stated in [3], defense strength, system performance degradation, scalability, implementation complexity, and compromise-ability of a defense system are very critical for a DDoS defense system. In addition, a Network/Transport-Level DDoS defense mechanism should handle high traffic injection rates; therefore, the allowable malicious traffic injection speed should also be considered during evaluation.

5.2.1 Defense Strength

Defense strength of a system demonstrates the detection and prevention performance of it against malicious activities. According to the responses of defense systems, four different outcomes can be obtained; false positive, false negative, true positive and true negative, as explained in ***Table 5.2*** [3].

Table 5.2: Possible Outcomes of a Defense System

	Desired Outcome is Positive	***Desired Outcome is Negative***
System Response is Positive	True Positive (TP)	False Positive (FP)
System Response is Negative	False Negative (FN)	True Negative (TN)

Six defense strength performance metrics can be calculated based on the outcomes of the defense system and these metrics can be used to evaluate a defense system. These metrics are calculated as shown in the ***Table 5.3*** [3].

Table 5.3: Defense Strength Performance Metrics

METRIC TYPE	***EXPLANATION***	***MEASUREMENT***
Accuracy	Measures the degree of the correct responses of a defense system to all events	$\text{ACC} = \frac{(\text{TP} + \text{TN})}{(\text{FP} + \text{FN} + \text{TP} + \text{TN})}$
Sensitivity	Measures the degree of the accurate positive responses over all positive events	$\text{SENS} = (\text{TP}) / (\text{TP} + \text{FN})$
Specificity	measures the degree of the accurate negative responses over all negative events	$\text{SPEC} = (\text{TN}) / (\text{FP} + \text{TN})$
Precision	measures the degree of the accurate positive responses over all positive responses	$\text{PREC} = (\text{TP}) / (\text{TP} + \text{FP})$
False Positive Rate	measures the degree of the inaccurate positive responses over all positive responses	$\text{FPR} = (\text{FP}) / (\text{TP} + \text{FP})$
False Negative Rate	measures the degree of the inaccurate negative responses over all negative responses	$\text{FNR} = (\text{FN}) / (\text{TN} + \text{FN})$

During the evaluation of the MiddleModule system, Sensitivity, Accuracy and False Positive Rate values are calculated and analyzed.

5.2.2 Scalability

Scalability is a critical evaluation metric in networking since the amount of traffic passing through a network system and the number of nodes connected to a network system can increase or decrease, and the system performance should not be affected dramatically with such changes.

5.2.3 System Performance Degradation

System Performance Degradation metric measures the performance degradation experienced in the underlying network that is caused by employing the defense system. The implemented defense system may increase the workload of the network nodes or may increase the forwarding delay of the nodes; and these may cause overall system performance to degrade. This is a critical issue especially for a data plane-based DDoS defense system because the defense system is implemented on the forwarding devices. Hence, the forwarding delay, or the average packet retrieval delay (in seconds) caused by the additional defense system on the legitimate hosts should be analyzed carefully. For that purpose, the network should be tested with and without the proposed system and packet retrieval times (time between sending packet request and receiving reply) of legitimate hosts are measured (in seconds).

5.2.4 Implementation Complexity

Implementation Complexity metric demonstrates the required additional resources and the required number of devices that should be modified to implement a defense system. When implementation complexity increases, the feasibility of implementing the defense system decreases.

5.2.5 Allowable Packet Receiving Rate

Allowable packet receiving rate is an important aspect for a data plane-based Network/Transport-Level DDoS defense mechanism. One of the most distinctive properties of the Network/Transport-Level DDoS attacks is that the attackers inject DDoS traffic into the network with the highest possible packet generation rate. In today's advanced networking technologies, the packet generation rate of single hosts is increasing; hence, the maximum allowable data rate (in packets per second) of a defense system becomes a critical performance metric. DDoS defense systems should be designed to handle higher packet generation rates and the defense strength performance of these systems should also be tested under such conditions.

5.2.6 Compromise-ability

All network defense systems are used to protect the network against some malicious activities. However, if the implemented network defense system can be exploited by the malicious users by some means then the network can be exposed to any network attack. This is a significant issue because when a defense system is exploited, the accuracy values or system performance degradation values calculated for that defense system becomes unimportant because the network is now defenseless to attacks under that condition. Therefore, during the evaluation of any defense system, the compromise-ability of the system should also be discussed in detail.

5.3 Evaluation Tests and Test Results

During the evaluation process of the MiddleModule system, several tests are applied to the system at different test platforms. For this purpose, OMNET++ 4.2 simulation platform and Mininet emulation platform are used, as was explained earlier.

Firstly, the system performance degradation of the proposed method is tested in OMNET++ platform. For this purpose, average packet retrieval time of a *Normal User* is measured with and without the MiddleModule system. Then the defense strength evaluation is performed in OMNET++ platform with custom test scenarios. The test scenarios used in defense strength tests are designed by considering the MiddleModule system working principles. After observing that the MiddleModule system functions properly against targeted malicious attack types, the detailed performance analysis tests are started in Mininet. In this platform, first the system performance degradation is measured; and then the defense strength is measured using different test scenarios and different packet generation rates. The test scenarios, the legitimate traffic generation datasets, and the targeted DDoS attack types that are used during the tests in Mininet platform are selected by considering similar studies found in the literature. The results collected in these tests are used to compare the performance of the MiddleModule system with existing works.

To increase the reliability and accuracy of the tests, test procedures are designed by considering the following points:

1. Generating Realistic Legitimate Traffic:

In the simulation tests in OMNET++ 4.2, the legitimate traffic is generated by using NSL-KDD dataset; and in the emulation tests, in Mininet, a real traffic dataset received from MAWI Working Group Traffic Archive is used. The legitimate traffic is generated by using these datasets and the malicious traffic is generated by using both the *malicious* packets in these datasets and obvious properties of the corresponding attacks. To increase the reliability of the emulation results, MAWILab traffic is used because it includes several non-malicious abnormalities, such as SYN bursts, packet errors, packet retransmissions, triple ACKs, etc. In Mininet, to inject both the legitimate and the malicious traffic into the network with different data rates, Scapy [52] and packETH [53] programs are used. Scapy is a sophisticated packet generation tool with many capabilities; however, it cannot inject traffic with rates higher than *1 pckt / 500ms*. On the other hand, packETH is a leightweigth packet generation tool and it can inject traffic with higher data rates up to *1 pckt / 1μs*. In addition, to fluctuate the legitimate traffic in both testing platforms, the legitimate traffic is built by using both TCP packets and ICMP packets in different percentages.

2. Applying Tests for Multiple Rounds:

During the evaluation, to increase the reliability of the tests, each test is applied for multiple rounds, instead of just one round. Each round lasts for a period of time or lasts until each host sends an exact number of packets. The average performance values for these rounds are calculated and reported as the result of the corresponding tests. In each test, the details about how the method is implemented for that specific test are explained.

3. Generating Traffic with Different Data Rates:

Different traffic generation speeds are used for both legitimate and malicious traffic during the evaluation; namely ultra-low-speed, low-speed, normal-speed, high-speed and the ultra-high-speed modes according to ***Table 5.4***:

Table 5.4: Traffic Generation Modes

Traffic Generation Speed Mode	***Packet Generation Rate***	***The Rate of Transferred Data*** *when the average packet size is 250 Bytes*
ultra-low-speed mode	*100 packets/sec, 1 pckt / 10 ms*	*200 Kbps*
low-speed mode	*500 packets/sec, 1 pckt / 2 ms*	*1 Mbps*
normal-speed mode	*5000 packets/sec, 1 pckt / 200 µs*	*10 Mbps*
high-speed mode	*100Kpackets/sec, 1 pckt / 10 µs*	*200Mbps*
ultra-high-speed mode	*500Kpackets/sec, 1 pckt / 2 µs*	*1Gbps*

5.3.1 Evaluation in OMNET++ 4.2

5.3.1.1 System Performance Degradation Test

The test scenario shown in ***Figure 5.1*** is built in OMNET++ 4.2 platform. In this test, the effect of the MiddleModule system on average packet retrieval time of the *Normal User* is measured. The test is applied using four different scenarios, with and without MiddleModule system and with and without the malicious traffic generated by the *Malicious User* nodes. Firstly, the *Normal User* generates legitimate TCP traffic towards the *Server* by using NSL-KDD dataset, while ten *Malicious User* hosts do not generate any traffic, and the MiddleModule system is deactivated.

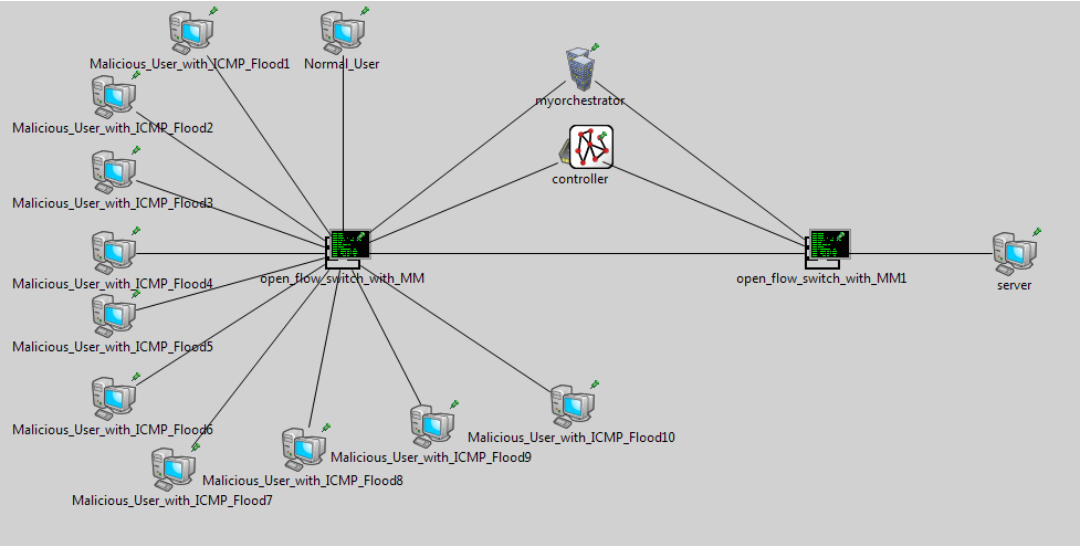


Figure 5.1: System Performance Degradation Test Scenario for OMNET++

During this test, the following steps, explained in ***Table 5.5***, are used.

Table 5.5: Test Steps in OMNET++

1. Generate the appropriate network topology in '*test.ned*' file.

2. Set the following properties in '*omnetpp.ini*' file for *Normal_User* node, by using the modified version of *TCPBasicClientApp*. Therefore, *Normal_User* node will operate as TCP client and generates packets according to NSL-KDD dataset.
   a. ***.Normal_User.numTcpApps = 1*
   b. ***.Normal_User.tcpApp[*].typename = "TCPBasicClientApp"*
   c. ***.Normal_User.tcpApp[*].kMeansMode = "normal"*
   d. ***.Normal_User.tcpApp[*].connectAddress = "server"*

3. Set the following properties in 'omnetpp.ini' file for each Malicious_User node,
   a. ***.Malicious_User*. numPingApps = 1*
   b. ***.Malicious_User*.pingApp[*].isIPSpoofing = true*
   c. ***.Malicious_User*.pingApp[*].typename = "PingApp"*
   d. ***.Malicious_User*.pingApp[*].destAddr = "server"*
   e. ***.Malicious_User*.pingApp[*].packetSize = 250B*
   f. ***.Malicious_User*.pingApp[*].count = 10000*
   g. ***.Malicious_User*.pingApp[*].sendInterval = 10us*

4. Set the following properties in '*omnetpp.ini*' file for each *open_flow_switch* node,
   a. ***.open_flow_switch*.open_Flow_Processing.windowSize = 40*
   b. ***.open_flow_switch*.open_Flow_Processing.saturationThreshold = 0.15*
   c. ***.open_flow_switch*.open_Flow_Processing.maliciousThreshold = 0.8*

5. Set the following properties in 'omnetpp.ini' file for each server node, by using the modified version of TCPEchoApp. Therefore, server node will operate as both TCP and ICMP web server. TCP web server generate responses according to NSL-KDD dataset.
   a. ***.server*.numTcpApps = 1*
   b. ***.server*.numPingApps = 1*
   c. ***.server*.pingApp[*].typename = "PingApp"*
   d. ***.server*.tcpApp[*].typename = "TCPEchoApp"*
   e. ***.server*.tcpApp[0].localAddress = ""*
   f. ***.server*.tcpApp[0].localPort = 1000*
   g. ***.server*.tcpApp[0].echoFactor = 1.0*
   h. ***.server*.tcpApp[0].echoDelay = 0*

6. Comment out the entries for *Malicious_User* nodes, set the following property in '*omnetpp.ini*', and apply test. Therefore, *Malicious_User* nodes do not generate any packets and MiddleModule system does not operate.
   a. ***.open_flow_switch*.open_Flow_Processing.isMMIDSOn = false*

7. Under these conditions, *Normal_User* node establishes new TCP connection, then generates TCP packets by using NSL-KDD. The modified version of *TCPBasicClientApp* writes the SYN request generation time, FIN reply reception time, and their differences on the simulation window, in *milliseconds*.

First, apply the steps from 1 to 7. In this test, a web server is built in the *server* node and the *Normal User* node creates a TCP connection to the *server* node. After

establishing the connection, *Normal User* node exchanges data with the server and then closes the TCP connection. The difference between connection initialization time and FIN packet reception time by the *Normal User* node is shown in *milliseconds* on the simulator window. This time difference shows single packet retrieval time of a *Normal User* node without DDoS attacks and without MiddleModule system. Apply this test for 10 rounds, where each round includes 100 connections. Calculate the average packet retrieval time in *milliseconds*.

In the second test, apply the steps from 1 to 7, but in step 6, set the parameter to *true*, as shown below.

***.open_flow_switch*.open_Flow_Processing.isMMIDSOn = true*

In this test, the *Malicious User* nodes do not generate any traffic but the MiddleModule system is activated. Follow the test procedure of the first test and calculate the average packet retrieval times for 10 rounds in *milliseconds*.

In the third and fourth tests, the previous two tests are repeated while *Malicious User* nodes are generating IP Spoofed SYN Flood traffic at *high-speed mode*. For this purpose, in step 5, comment in the entries for *Malicious User* nodes. Calculate the overall average packet retrieval times under these conditions in *milliseconds*. Obviously, the packet retrieval times are in simulation time, not in real time.

The overall average packet retrieval times are compared in ***Table 5.6***. As this table demonstrates, the MiddleModule system does not cause any noticeable degradation at *Normal User* packet retrieval time when there are no DDoS attacks. When the *Malicious User* nodes generate DDoS traffic, because of switch buffer overflows and switch flow table overflows, mentioned earlier, the *Normal User* cannot get any response from the *server* if the MiddleModule system is deactivated. Average packet retrieval time of the *Normal User* goes to infinity. On the other hand, if the MiddleModule system operates during DDoS attacks, the *Normal User* can communicate with the *server* without any packet loss, and the average packet retrieval time is very close to the average packet retrieval time when there is no DDoS attack on the network.

Table 5.6: System Performance Degradation Result in OMNET++

	without DDoS traffic	**with DDoS traffic**
without MM	***0.531 ms***	∞
with MM	***0.532 ms***	***0.534 ms***

5.3.1.2 Defense Strength Test

In OMNET++ 4.2 simulation platform, the defense strength tests are applied to the MiddleModule system to satisfy three goals. Firstly, the tests are applied for basic functional analysis to observe if the MiddleModule system detects the targeted malicious activities, or not. Secondly, further functional analysis is performed to observe if the MiddleModule system operates correctly under more complex test scenarios. To satisfy the second goal, all targeted Network/Transport-Level DDoS attacks are generated by the *Malicious User* nodes at the same time, and the legitimate traffic is generated by the *Normal User*. Under these conditions, the detection performance of the MiddleModule system is evaluated. Finally, the configuration variables of the MiddleModule system, *detection algorithm thresholds* and *windowSize*, are varied and the defense strength performance of the MiddleModule system is measured. The relation between the defense strength performance of the MiddleModule system and the configuration variables are evaluated with this test. In these tests, IP spoofing attacks, SYN Flood attacks, ICMP Flood attacks, UDP Flood attacks, ICMP Reflection attacks, and Broadcast Amplification attacks are targeted.

During these tests, the steps from 1 to 7, explained in ***Table 5.5*** are used. The step 3 is modified according to the applied test. For example, for UDP Flooding, *"UDPApp"* is used instead of *"PingApp"*. In step 5, UDP server application is also operated, and in step 6, comment in the entries for *Malicious User* nodes and set the parameter to *true*, as shown below. During detection algorithm parameter dependency analysis of the MiddleModule system, the parameters in step 4 are varied according to ***Table 5.8***. The detection and prevention decision are shown in

the simulation window thanks to the modifications in '*open_flow_switch'* source code.

***.open_flow_switch*.open_Flow_Processing.isMMIDSOn = true*

First, the test scenario, shown in the ***Figure 5.2***, is created in the OMNET++ platform to test the basic functionality of the MiddleModule system. The DDoS detection functions are built in edge-switches, *open_flow_switch_withMM* and *open_flow_switch_withMM1*, and they are connected to *myorchestrator* module, which simulates the proposed Orchestrator Block residing in the Controller. If *open_flow_switch_withMM* module detects any malicious activity, it informs *myorchestrator* module. Five different tests are applied and in each one of them different malicious traffic (IP Spoofing traffic, SYN Flood traffic, ICMP Flood traffic, UDP Flood traffic and Broadcast Amplification traffic) is generated from the *Malicious User* module towards the *server* module. In these tests, *Malicious User* generates traffic at *high-speed mode* and the configuration variables are set according to ***Table 5.9***.

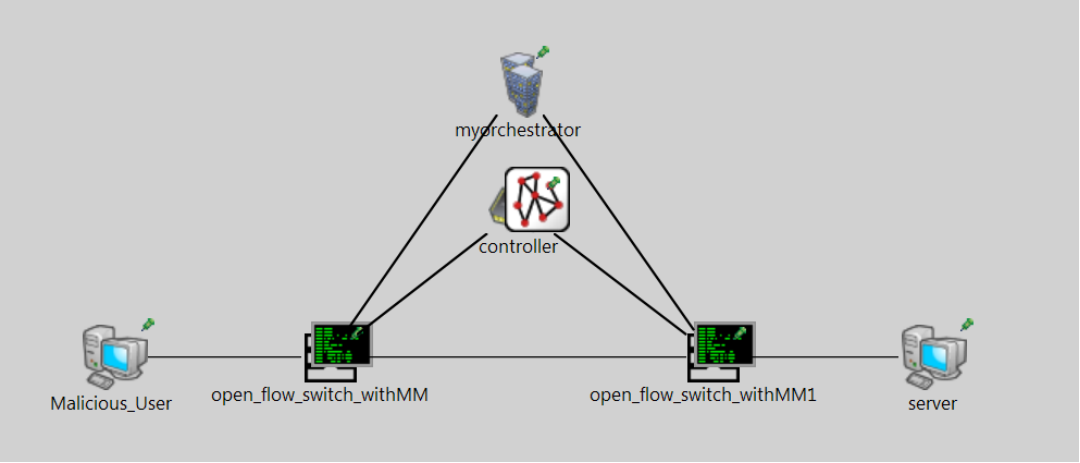


Figure 5.2: Basic Defense Strength Test Scenario for OMNET++

Test results show that, malicious packets are detected by the *open_flow_switch_withMM* module when they satisfy the malicious conditions. Since the *open_flow_switch_withMM* module is closer to the source of the malicious activity, this module detects and prevents the malicious activity. The prevention

decisions of *open_flow_switch_withMM* node are analyzed, and the detection rate is calculated as follows;

$$Detection\ rate = \frac{\text{prevented Malicious Packets count}}{\text{all Malicious Packets count}}$$

Under these conditions, the detection rate of the MiddleModule system is measured to be above 99%. Some of the detection and prevention operation outputs, collected from the OMNET++ simulation platform, are shown in the ***Figure 5.3***.

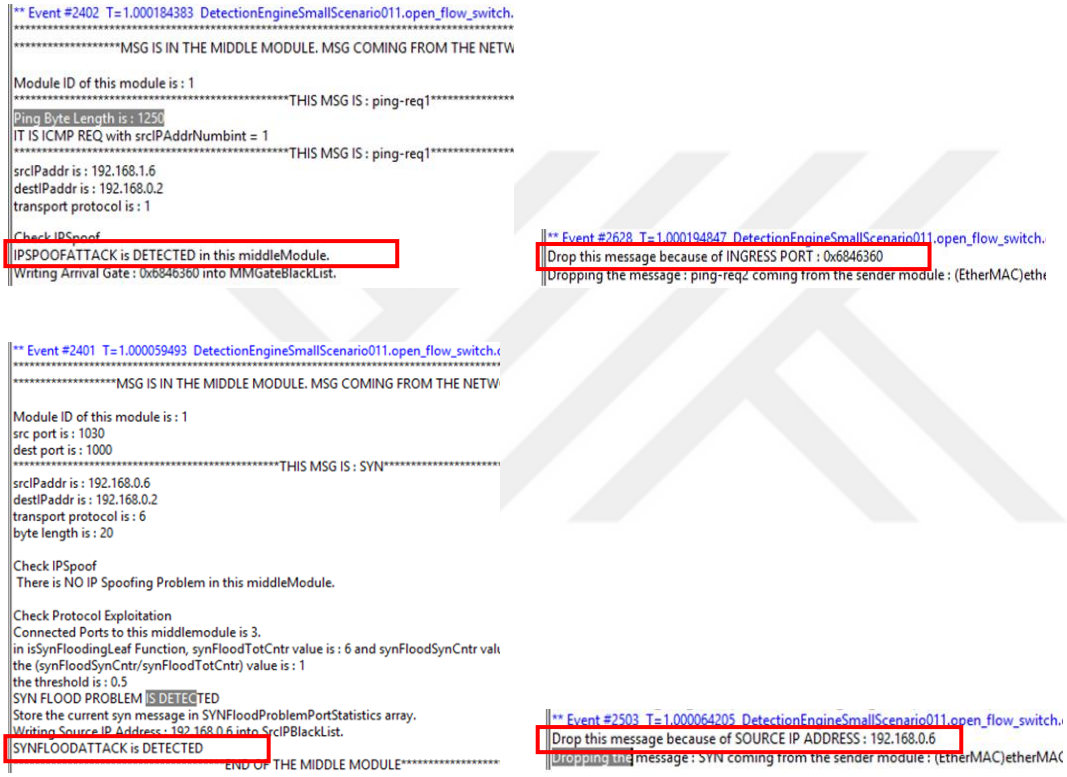


Figure 5.3: Basic Defense Strength Test Results from OMNET++

A second but a more complex test scenario is built in the OMNET++ simulation and the defense strength tests are reapplied using this scenario. During this test, both legitimate traffic and the malicious traffic are generated at the same time. Different attack types are also generated by different *Malicious User* nodes. For this test, the scenario shown in ***Figure 5.4*** is built in OMNET++, and legitimate traffic is generated by *Normal User* and *Normal User1* modules. To increase the reliability of the test, the legitimate traffic is generated by using two different communication

protocols, TCP and ICMP, and 90 percent of the legitimate traffic is built with TCP while the remaining 10 percent is built with ICMP. The TCP traffic is generated by using NSL-KDD dataset and the ICMP traffic is generated as *ping* messages from *Normal User* modules towards *server* and *server1* modules. As demonstrated in ***Figure 5.4***, three *Malicious User* nodes generate ICMP Reflection attack towards the *server* module, one *Malicious User* node generates Broadcast Amplification attack towards *server1* module, and *Normal User* module generates legitimate traffic towards *server* module while *Normal User1* module generates legitimate traffic towards *server1* module. In addition, one *Malicious User* generates UDP Flooding attack towards to *server* module, while one *Malicious User* generates SYN Flooding attack with IP Spoof towards *server1* module. The configuration variables of the detection algorithms implemented in edge-switches are set according to ***Table 5.9***. Malicious traffic is generated at *high-speed mode* for 10 rounds, and each round lasts for 20 msec simulation time.

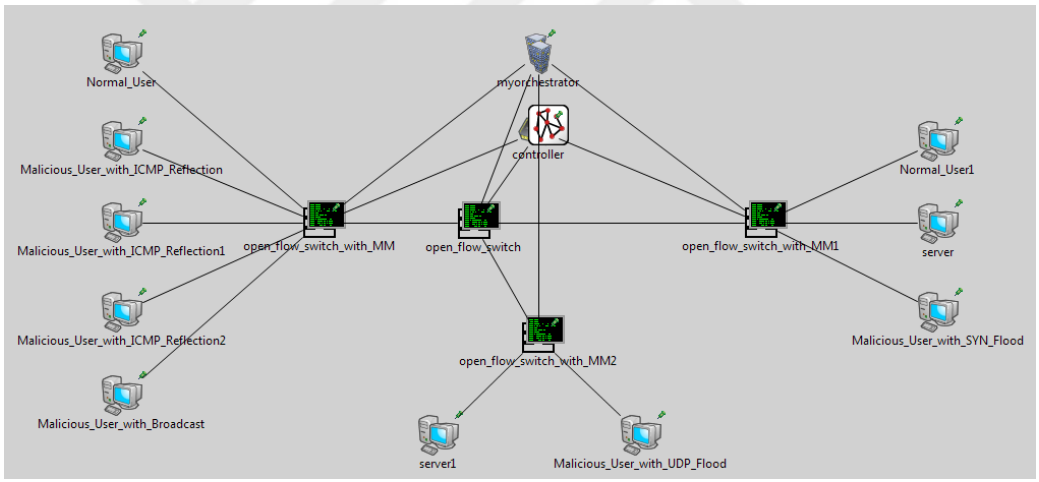


Figure 5.4: Complex Defense Strength Test Scenario for OMNET++

The prevention decisions of *open_flow_switch_withMM* node are analyzed, and the detection rate (sensitivity) and false positive rates are calculated as follows;

$$Detection\ rate = \frac{\text{prevented Malicious Packets count}}{\text{all Malicious Packets count}}$$

$$False\ Positive\ rate = \frac{\text{prevented Normal Packets count}}{\text{all Normal Packets count}}$$

Simulation results demonstrate that the MiddleModule system detects the malicious activities accurately, while forwarding the legitimate traffic without any packet loss. Hence, during DDoS attacks, thanks to the MiddleModule system, the *Normal User* nodes can communicate with the *server* nodes successfully while the traffic coming from *Malicious User* nodes are prevented at the data plane devices, without sending them to *server* nodes or sending them to controller. Under these conditions, the sensitivity, false positive rate and accuracy of the MiddleModule system model are measured; and they are shown in ***Table 5.7***. The sensitivity results are measured to be above 99.6 % for all attack types, and the accuracy results are above 99.8 %; which means all targeted Network/Transport-Level DDoS attack types are detected accurately by the MiddleModule system. On the other hand, the legitimate packets are not dropped by the MiddleModule system; hence, the false positive rate is zero.

Table 5.7: Simulation Defense Strength Performance Results

	Sensitivity	***False Positive Rate***	***Accuracy***
IP Spoofing Attack	99.8 %	0 %	99.9 %
SYN Flooding Attack	99.6 %	0 %	99.8 %
UDP Flooding Attack	99.6 %	0 %	99.8 %
ICMP Reflection Attack	99.7 %	0 %	99.8 %
Broadcast Amplification Attack	99.7 %	0 %	99.8 %

Finally, the defense strength performance variation of the MiddleModule system is evaluated with respect to three configuration variables, namely *saturationThreshold, maliciousThreshold* and *windowSize*. For this purpose, the test scenario, shown in ***Figure 5.5*** is built in OMNET++. During *saturationThreshold* and *maliciousThreshold* evaluations, *Malicious User* nodes generate only SYN Flood traffic without IP Spoofing. In these tests, *windowSize* is assigned to an expected optimum value of 20. Then, other parameters are set to their optimum values and the effect of *windowSize* is evaluated with SYN Flood traffic. Finally, the *Malicious User* nodes generate ICMP Reflection attack, and the effect of *windowSize* is evaluated for this attack type, also. Tests are applied as 10 rounds and, in each round,

each host node, both *Malicious User* and *Normal User*, generates 200 packets towards the *server* node.

The legitimate traffic is built using the TCP protocol. Different TCP connections; including 3-way handshaking, data transfer, and FIN/ACK message exchange; are created by using the NSL-KDD dataset. If a non-zero payload length is provided in the dataset for a *benign* connection, then such data transfer occurs between the *Normal User* node and the *server* node for that particular connection. If payload length provided in the dataset for a *benign* connection is zero, then no data transfer occurs between the nodes in that connection; only the 3-way handshaking and FIN/ACK message exchange occur. For each scenario, demonstrated in ***Table 5.8***, test is applied for 10 rounds. The defense strength outcomes are measured for each round and average accuracy and false positive rates are calculated. As shown in ***Table 5.8***, different values for the associated variables are selected from different intervals. As a result of this test, the relation between the defense strength of the MiddleModule system and configuration variables are evaluated; and ideal configuration variable values, for the network topology in ***Figure 5.5*** are found.

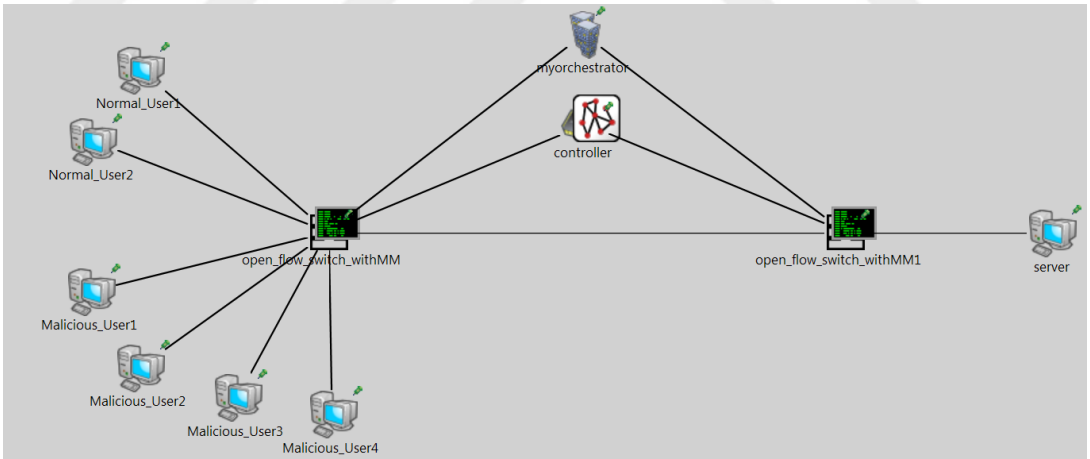


Figure 5.5: Defense Strength Variation with Configuration Variables Test Scenario

Table 5.8: Configuration Variable Variation

	synFlood Threshold (maliciousThreshold)	***saturation Threshold***	***windowSize***	***Malicious Traffic Type***
Test 1	*0.8*	*0.15*	*20*	*SYN Flood*
Test 2	*0.8*	*0.75*	*20*	*SYN Flood*
Test 3	*0.2*	*0.25*	*20*	*SYN Flood*
Test 4	*0.4*	*0.25*	*20*	*SYN Flood*
Test 5	*0.5*	*0.25*	*20*	*SYN Flood*
Test 6	*0.7*	*0.25*	*20*	*SYN Flood*
Test 7	*0.99*	*0.25*	*20*	*SYN Flood*
Test 8	*0.8*	*0.25*	*5*	*SYN Flood*
Test 9	*0.8*	*0.25*	*20*	*SYN Flood*
Test 10	*0.8*	*0.25*	*100*	*SYN Flood*
Test 11	*0.8*	*0.25*	*5*	*ICMP Reflection*
Test 12	*0.8*	*0.25*	*20*	*ICMP Reflection*
Test 13	*0.8*	*0.25*	*100*	*ICMP Reflection*

As shown in ***Figure 5.6***, the defense strength of the MiddleModule system changes with respect to configuration variables. The following three conclusions can be drawn about configuration variables:

First, ***synFloodThreshold*** should not be assigned a low value to avoid high false positive rates. If this variable is set to a high value, near 1.0 for example, then the detection rate of the system can decrease if any unexpected change occurs in the malicious traffic. Although, test results do not highlight this point, setting this variable too high is also not advised considered the structure of the detection algorithms. Hence, ***synFloodThreshold*** variable can be set a value from moderate to high interval. Second, the ***saturationThreshold*** is used to prevent immature decisions; therefore, it should not be set too low. This variable should not be set too high also. The detection algorithms wait until they receive a sufficient number of packets to satisfy the saturation condition before making any decisions. If this variable is set too high, the MiddleModule system misses many malicious packets. Therefore, it should be set to a value from moderate to low interval. Finally, the variable ***windowSize*** should be chosen according to the network topology. If it is set

too low for a given topology, then the accuracy of the detection algorithms decreases. If it is set too high, then the detection algorithms cannot analyze the most recent packets, which decreases the detection performance of the system. In addition to these outcomes, there are two important conclusions drawn from our tests; accuracy and false positive performances changes steadily near optimum points; therefore, it is not vital to use the exact optimum values. Setting the configuration variables close to optimum points is enough to get high performance results. Secondly, defense strength changes with the configuration variables, in a predictable way. Hence, the optimum values for these variables can be predicted by considering the network topology. In this test, the optimum values for the given network topology are found as shown with green dots in ***Figure 5.6***.

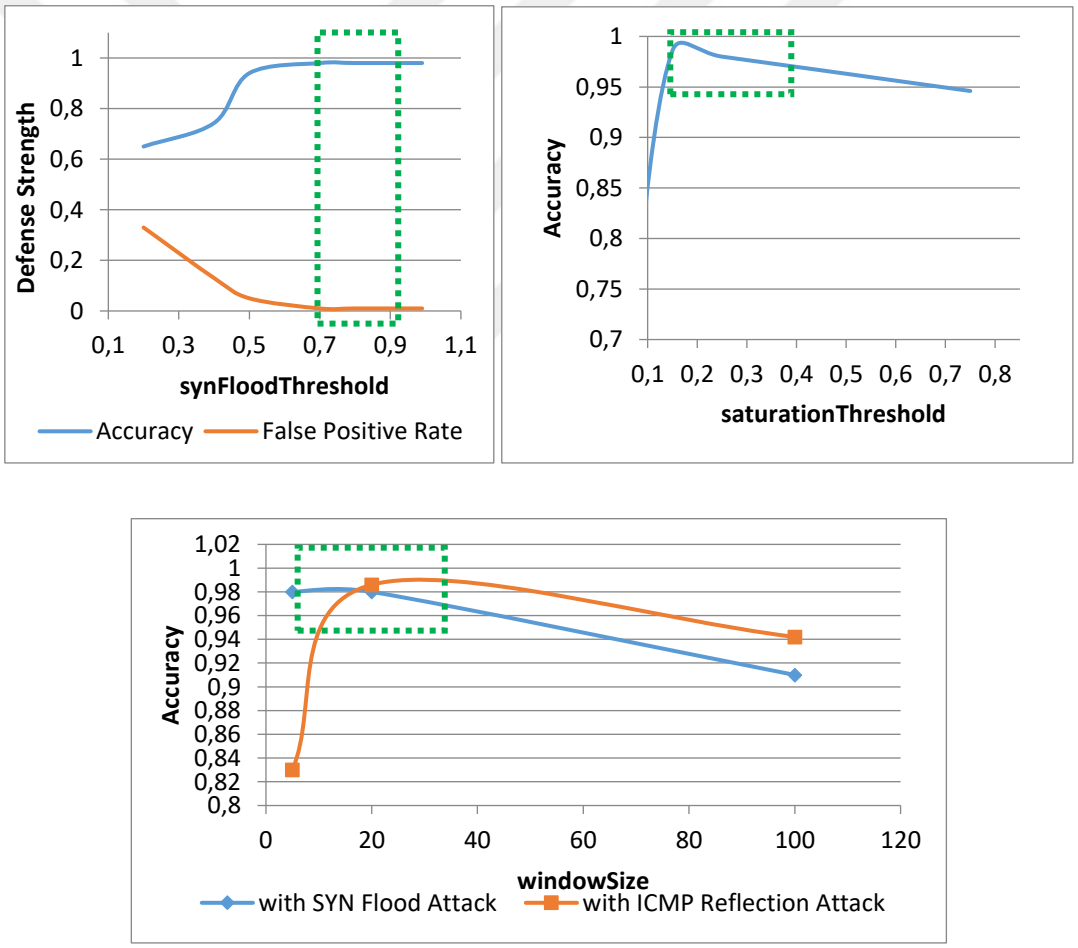


Figure 5.6: Defense Strength Variation Results with respect to Configuration Variables

During the emulation tests performed in Mininet platform, a test scenario, similar to the one demonstrated in ***Figure 5.5***, is used. Therefore, to obtain the best performance in emulation tests, configuration variables are set as shown in ***Table 5.9***.

Table 5.9: Optimum Configuration Variables for the Test Scenario

windowSize	20 - 40
saturationThreshold	0.15 - 0.30
spoofThreshold	2
synFloodThreshold	0.70 – 0.90
floodThreshold	0.70 – 0.90

5.3.2 Evaluation in Mininet

Although OMNET++ provides realistic test results, simulators have some limitations in terms of reliability, as was mentioned earlier. To provide more reliable and more accurate results, some of the performance tests are repeated in Mininet platform. These tests are applied to detect system performance degradation and defense strength of the MiddleModule system using different packet generation rates. Scalability of the system is also discussed. In addition, the results collected in Mininet are used for performance comparison of the proposed system with similar studies. In our tests, Mininet is built on top of VirtualBox by using Mininet VM image provided in [54]. The virtual machine that runs the Mininet has the properties shown in ***Table 5.10***. The Mininet version 2.2.2 and OpenFlow version 1.0 is used in these tests. In the test scenarios, Open vSwitch 2.0.2 and OVS-Controller 2.0.2 are used as the OpenFlow compliant network nodes.

Table 5.10: VirtualBox Properties

VirtualBox version	5.2.6 r120293
Operating System	Ubuntu – 64bit
Number of Cores	1
Installed Memory	6 GB
Upper Limit of Operation	100 %

These tests are applied by following the steps explained in ***Table 5.11***.

Table 5.11: Test Steps in Mininet

1.	Run Mininet-VM (Mininet virtual machine) in VirtualBox, then run Mininet GUI/X11 by typing '*startx*' on the command line.
2.	Go to '*~/mininet/custom*' folder and build the required network topology with the name '*myTopology.py*' for the test.
3.	In the terminal command line, type '*sudo mn --custom custom/myTopology.py --topo myTopo --link tc -x*'. This will start the mininet emulation for the network, defined in '*myTopology.py*' file. In addition, for each network node, *xterm* command lines will open.
4.	At the *switch* node, read the incoming traffic with *tcpdump* command and then run the MiddleModule functions on the received packets.
5.	Type '*python -m SimpleHTTPServer 80 &'* to the command line of *server* node to create a simple HTTP server with the port number of 80.
6.	Type '*wget -0 -destIPAddr &*' to the command line of *Normal User* node to get packet from the HTTP server node.
7.	To generate legitimate TCP traffic, use *scapy*. Read the '*.pcap' file of TCP traffic dataset with *rdpcap* command and modify the source and destination header information of each packet according to the test network. Then send the packet with *sendp* command. To generate legitimate ICMP traffic, again the *rdpcap* and *sendp* commands can be used.
8.	To generate reply messages to the legitimate TCP packets, use *scapy* again with the corresponding '*.pcap' file.
9.	To generate malicious traffic at *Malicious User* node, if the packet generation rate is less than *1 pckt/500 ms*, use scapy. Read the malicious traffic dataset with *rdpcap*, modify the packet headers appropriately then send the packets with *sendp*. If generation rate is higher than *1 pckt/500 ms*, use packETH. Type '*./packETHcli -i eth0 -m 2 -d delayBetweenPacketInus -n numberOfPackets -f maliciousTraffic.pcap*' to the command line of the *Malicious User* node.
10.	At packet generation or packet reception at all nodes, use also the *strftime* function to read time of that action in *milliseconds*.
11.	At the *server* node, run *tcpdump* to observe incoming traffic.

5.3.2.1 System Performance Degradation Test

In the Mininet platform, the test scenario shown in ***Figure 5.7*** is built and an HTTP web server application is operated at the *Server* node. First, the MiddleModule is deactivated and under this condition, *Normal User* connects to the *Server* and receives packets from the HTTP web server, and the average packet retrieval time is measured. This test is applied for 10 rounds, each one including 100 packet retrievals, and the average packet retrieval time is calculated for each round and also for the overall test. Then the MiddleModule is activated and *Normal User* connects to *Server*. The packet retrieval times are again measured for 10 rounds, and 100 packet retrievals for each round. The average packet retrieval time of the *Normal User* are calculated for each round under these conditions.

During this test, follow the steps 1, 2, 3, 4, 5, 6, 10 explained in ***Table 5.11***. To deactivate the MiddleModule system, at step 4, do not use any MiddleModule function. Compare the times, received at step 10, for packet request generations and packet response receptions at the *Normal User* node, and calculate the packet retrieval times.

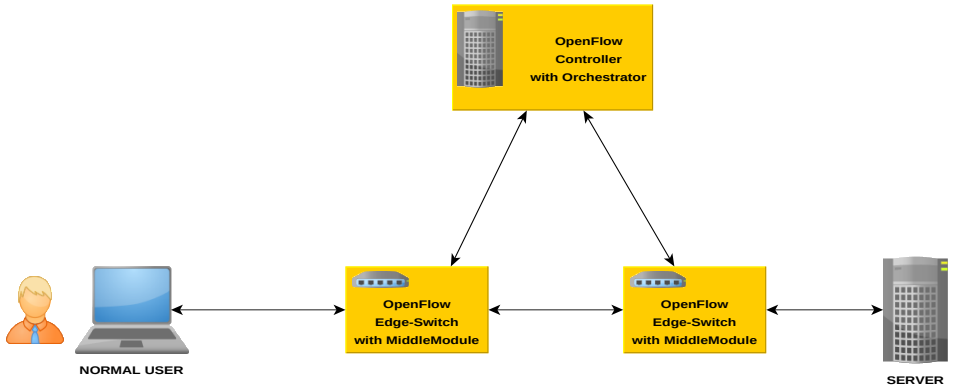


Figure 5.7: System Performance Degradation Test Scenario for Mininet

Table 5.12 illustrates that the average packet retrieval times, in *seconds*, with and without the MiddleModule are very close to each other, and the packet retrieval time overhead caused by the MiddleModule system is *0.2%*. Hence, the system

performance degradation of the MiddleModule system in terms of packet retrieval time is negligible.

Table 5.12: System Performance Degradation Result in Mininet

	without MiddleModule (in sec)	***with MiddleModule (in sec)***
Round1	0.02530	0.02532
Round2	0.02516	0.02531
Round3	0.02531	0.02526
Round4	0.02542	0.02544
Round5	0.02534	0.02532
Round6	0.02542	0.02528
Round7	0.02528	0.02556
Round8	0.02501	0.02547
Round9	0.02533	0.02503
Round10	0.02556	0.02520
Average	***0.025313***	***0.025319***
Overhead	-	***0.2 %***

5.3.2.2 Defense Strength Test

In Mininet, the accuracy and the false positive rate of the MiddleModule system are measured against IP Spoofing attack, IP spoofed/non-spoofed SYN Flood attack, IP spoofed/non-spoofed ICMP Flood attack and IP spoofed/non-spoofed UDP Flood attack. The measurement results are used for comparing our MiddleModule system with similar studies found in the literature. Since there are no results for the reflection attacks and the broadcast amplification attacks provided in these studies, these attack types are not considered during the defense strength tests in Mininet. In these tests, scenarios are chosen to be similar to the ones used in the other studies. The configuration variables are chosen as in Table 5.9: Optimum Configuration Variables for the Test Scenario***Table 5.9*** to obtain ideal results. In addition, *penaltyTime* is set to 5 seconds. Tests are applied for 10 rounds and each round lasts 100 seconds. The accuracy and false positive rates are measured in each round; and the average of the measurement results is computed using 10 rounds. In these tests,

as was explained previously, the legitimate traffic is generated by using MAWILab dataset.

During this test, follow the steps 1, 2, 3, 4, 7, 8, 9, 10, 11 explained in ***Table 5.11***. For defense strength measurement, observe the generated packets by *Malicious User* nodes and *Normal User* nodes and also the received packets by *server* node, at the step 11. Measure the true positive, true negative, false positive and false negative values, by using this observation.

First, the test scenario shown in ***Figure 5.8*****Error! Reference source not found.** is built and different tests are applied to observe that the MiddleModule system detects and prevents malicious traffic correctly. In each test, a different DDoS attack type is generated by the *Malicious User* and the accuracy of the MiddleModule system is measured.

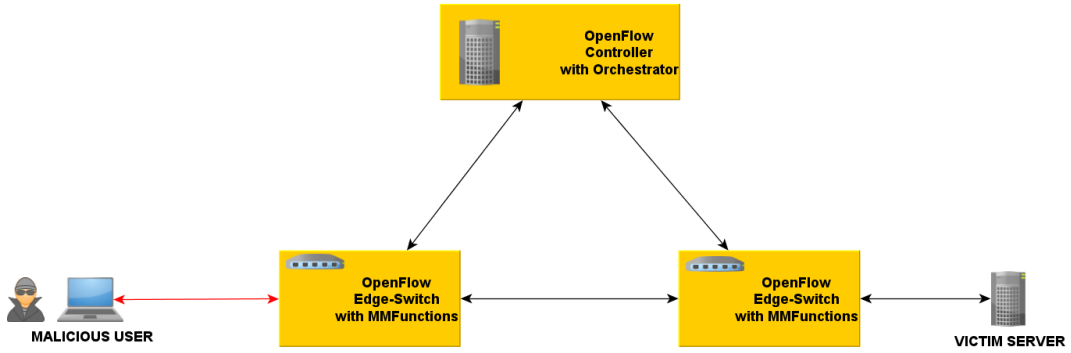


Figure 5.8: Test Scenario for Basic Defense Strength Tests

In the first test, the IP Spoofed ping messages (without Flooding) are generated from *Malicious User* towards the *Victim Server*. Malicious traffic is generated at *ultra-low-speed-mode* (*1 pckt / 10 ms*), and as shown in ***Figure 5.9***, the IP Spoof attack is detected when the received packet count exceeds the *saturationThreshold.* Then, the packets coming from the *Malicious Host* is prevented accurately.

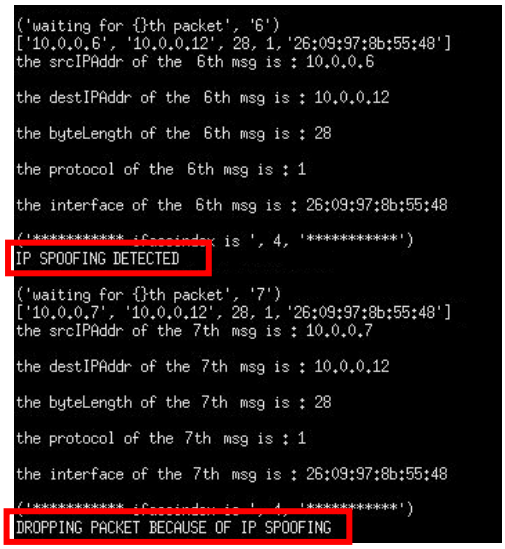


Figure 5.9: IP Spoof Attack Detection Result

Using the test scenario shown in ***Figure 5.8***, the SYN Flooding attack with IP Spoof and ICMP Flooding attack with IP Spoof and UDP Flooding attack with IP Spoof are generated from the *Malicious User* at *ultra-low-speed-mode (1 pckt / 10 ms)*, the MiddleModule system detects these attacks as IP Spoofing, successfully. After detecting the attack and labeling the ingress port of these messages as malicious, the packets received from that port is dropped, as can be seen in ***Figure 5.9.*** Finally, the tests with the remaining attack types, the SYN Flooding attack without IP Spoof, the ICMP Flooding attack without IP Spoof and the UDP Flooding attack without IP Spoof attacks are applied, at *ultra-low-speed mode*; and the MiddleModule system detected the attacks accurately. These tests are applied using other traffic generation rates also and it is observed that the detection performance of the system does not change even the test conditions are changed. The detection outcomes and the detection rate values are measured and shown in ***Table 5.13***. Since *penalty*Time value is set to 5sec, each test duration is considered to be 5sec. Hence, in each round, that lasts 100sec, 20 different sub-rounds are performed. The False Negative counts increase as the packet generation rate increases, because the testing tools add a certain delay to the detection system, and during this time, some of the malicious packets are missed. These detection delay values would get dramatically smaller if this system is implemented in an appropriate hardware platform. However, even with

such delay effects, the sensitivity value is remarkably high; i.e., it is above 99.5% for all traffic generation speeds.

Table 5.13: Detection Performance Results with Basic Test Scenario

	TP	***FN***	***Sensitivity***
low-speed-mode (1 pckt / 2 ms)	2489	11	99.5 %
normal-speed-mode (1 pckt / 200 µs)	24900	100	99.6 %
high-speed-mode (1 pckt / 10 µs)	498286	1714	99.6 %
ultra-high-speed-mode (1 pckt / 2 µs)	2491886	8114	99.6 %

Following the basic detection rate analysis of the MiddleModule system, a further defense strength analysis is performed to obtain results for more reliable and realistic test scenarios. A more realistic test scenario, including several *Normal User* nodes and several *Malicious User* nodes, shown in ***Figure 5.10***, is built in Mininet. During these tests, the legitimate traffic is built by using TCP packets and ICMP packets. The ratio of these packet types to overall packets is changed during the tests to fluctuate the legitimate traffic. The TCP packet ratio changes between 80% and 100%, while the ICMP packets ratio changes between 20% and 0%. The legitimate TCP packets are produced by using the MAWILab dataset, as was explained earlier. The packets are injected to the network and directed towards the *Victim Server* with Scapy or packETH tools, depending on the traffic rate. During these tests, the *windowSize* variable is set to 100 and the *saturationThreshold* variable is set to 0.25, to minimize false positive rates. *The spoofThreshold* is set to 2, *synFloodThreshold* and *floodThreshold* are set to 0.80, and *penaltyTime* is set to 5 seconds.

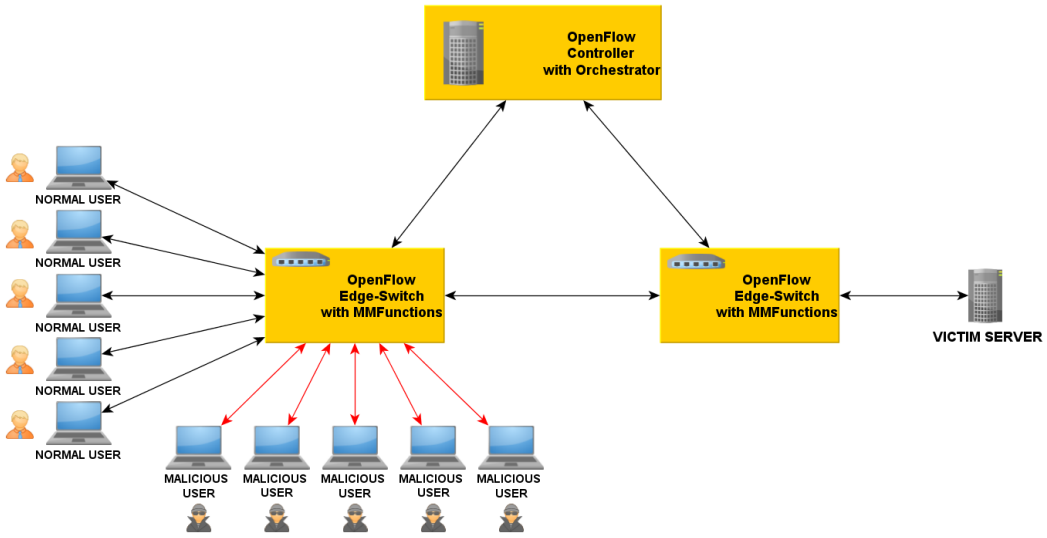


Figure 5.10: Test Scenario for Reliable Defense Strength Tests

In these tests, the following attack types are generated in that order; the IP Spoofing attack traffic, the SYN Flooding attack traffic with IP Spoofing, SYN Flooding attack traffic without IP Spoofing, the ICMP Flooding attack traffic with IP Spoofing, ICMP Flooding attack traffic without IP Spoofing, the UDP Flooding attack traffic with IP Spoofing, UDP Flooding attack traffic without IP Spoofing. In one test, only one attack type is generated from all *Malicious User* nodes. While the *Malicious User* nodes generate such traffic, the *Normal User* nodes generate legitimate traffic, as was mentioned earlier. The sensitivity, the accuracy and the false positive rate of the MiddleModule system are measured against each attack type, for 10 rounds, where each round lasts for 100 seconds; and the average results of these 10 rounds are computed. In addition, each test is applied for five different packet generation rates to measure the packet generation rate dependency of the system. The accuracy and the false positive rates are recorded for each packet generation speed mode. Since the *penalty*Time value is set to 5sec, each test duration is considered to be 5sec. Hence, in each round, lasting 100sec, 20 different sub-rounds are performed. The test results for different attack types are very similar to each other and the average results for all attack types are shown in ***Table 5.14***.

Table 5.14: Defense Strength Test Results in Mininet

	TP	***FN***	***FP***	***TN***	***Sensitivity***	***Accuracy***
ultra-low-speed-mode (1 pckt/10 ms)	482	18	0	500	96.4 %	98.2 %
low-speed-mode (1 pckt/2 ms)	7409	91	0	7500	98.8 %	99.4 %
normal-speed-mode (1 pckt/200 µs)	124143	857	0	125000	99.3 %	99.6 %
high-speed-mode (1 pckt/10 µs)	2483051	16948	0	2500000	99.3 %	99.6 %
ultra-high-speed-mode (1 pckt/2 µs)	7422748	78252	0	7500000	98.9 %	99.5 %

As given in ***Table 5.14***, the MiddleModule system does not cause any legitimate packet drop, while it detects and prevents malicious traffic successfully, for each traffic rate. Since there is no legitimate packet drop, *False Positive Rate* of the MiddleModule system under these test conditions becomes zero. On the other hand, some False Negatives are observed, and the number of False Negatives increases as packet generation rate increases because of certain delays caused by testing tools mentioned earlier. As shown in ***Table 5.14,*** *Accuracy* for all conditions are closer to or higher than 99%. This test result suggests two important points; firstly, the defense strength of the MiddleModule system with real traffic dataset and with the associated test scenario is remarkable for targeted Network/Transport-Level DDoS Attack types. Secondly, the defense strength of this system does not change dramatically with the change of the packet generation rate, and this system successfully detects attacks even with the 0.5M packets per second data rate.

5.3.2.3 Communication Overhead Measurement

In MiddleModule system, each data plane MiddleModule node operates by itself. The nodes do not need any command or any additional information, coming from other nodes, to operate correctly. However, to inform the controller about the prevention rules, and to obtain the best mitigation performance, data plane MiddleModule nodes send two different messages. Firstly, when a data plane MiddleModule node detects any malicious activity, it sends *malicious information message* to the controller to inform the controller about this malicious activity. This message is sent when the malicious activity is detected for the first time. Secondly,

when a malicious activity is detected at the destination side of a message, this data plane MiddleModule node sends *malicious distribution message*, to the controller. When the controller receives that message, it distributes this message to the other data plane MiddleModule nodes. By using *malicious distribution messages*, when a packet is received from a previously labeled malicious host, that packet is mitigated at the closest edge-switch to the source of the malicious activity; hence, the malicious packets are prevented before they enter the network. Both messages are TCP messages, including 64 bytes for header portion and 60 bytes for data payload portion, leading 124 bytes for a single message. In addition, both are sent when the malicious activity is detected for the first time; they are not periodic messages.

The communication burden of the MiddleModule system to the entire network is calculated according to the test scenario, shown in ***Figure 5.10***. In this test scenario, in a single round, lasting 100 seconds, with penalty time of 5 seconds, an edge-switch receives DDoS messages from 5 different malicious hosts; therefore, the total size of the *malicious information message* in the network is 124bytes x 20 x 5 =12.1 KB. On the other hand, when the malicious hosts generate Reflection attack, the destination side edge-switch sends *malicious distribution messages* to the controller, with the total size of 124bytes x 20 = 2.42KB. These messages are distributed to the other edge-switch by the controller, which leads total *malicious distribution messages* overhead of 12.1 KB. Therefore, the total communication overhead of the MiddleModule system is 24.2 KB for this test scenario, while the total traffic passing through the network is in the order of GBs. Hence, the communication burden of the MiddleModule system is less than 0.1 %, which is negligible.

On the other hand, the MiddleModule system provides additional capabilities to the network operator, such as, modifying the detection variables during operation, receiving periodic status messages from the data plane MiddleModule nodes, sending prevention commands to the data plane MiddleModule nodes. Since using these capabilities is optional and they are not required for accurate defense operations, the overhead, caused by these operations, is not evaluated in this thesis.

5.3.3 Scalability Analysis

The scalability of the MiddleModule system is evaluated in terms of the detection performance scalability. This is evaluated with different numbers of connected hosts and with different traffic injection speeds. In addition, the scalability of the required hardware is also discussed.

The detection rate performance of the MiddleModule system scales with respect to the number of connected hosts to a single edge-switch and the traffic injection speed by those hosts. Since all packets are evaluated with the same detection algorithm set, this is an expected value. In addition, since the MiddleModule system suggests using a single Packet Processing Block to process all incoming packets, the False Negative Rate may change slightly as the incoming packet number increases; however, the sensitivity rate or the false positive rate do not change noticeably under such conditions. As suggested in the evaluation tests, applied with the test scenarios shown in ***Figure 5.8*** and ***Figure 5.10***; as the number of connected hosts increases from one to ten and as the incoming traffic rate is changed from *1 pckt / 10ms* to *1pckt / 1us*, the accuracy of the MiddleModule system stays above 99% while the false positive rate is nearly 0%. In addition, since each edge-switch operates by itself, as the number of edge-switch in the network increases, the performance of each edge-switch does not change.

On the other hand, the required hardware, to provide MiddleModule functionality to an edge-switch, changes as the number of ingress ports of that edge-switch change. The size of the Packet Mirror and Attack Prevention Block increases linearly according to the ingress port number, and the allocated resources increase at the Packet Processing Block linearly. The scalability of the allocated hardware is guaranteed by the design of the system.

5.3.4 Implementation Complexity Analysis

The MiddleModule system suggests placing monitoring, detection and mitigation blocks at the data plane devices of the SDN network. This system suggests

modification of the edge-switches in the network, while the core switches are used without any modification. All packets in the network are processed by the defense system, while only some of the data plane devices are modified, thanks to this thesis. The ratio of the number of modified switches to the number of all switches depends on the network topology. This ratio could be close to 1 or 0.5. In a tree network topology, one of the most common network topologies [55], the ratio of the number of edge-switches to the number of all switches is $(2^{n-1}) / (2^n-1)$; for example 16/31 for a 5 stage network.

The modification in the edge-switches, on the other hand, is not a complex or comprehensive modification. Since the MiddleModule system uses very lightweight detection algorithms, the required hardware resources to operate these algorithms are not excessive. The required processing power is also not much. Since only the header portion of the incoming packets is processed by the detection algorithms, the required memory and IO bandwidth are also small. Furthermore, the detection operations are handled very close to the responsible hosts and edge-switches can detect any change at the traffic generated by these hosts only by analyzing the most recent packets. Therefore, the required table sizes to store the monitoring results for each host are also small.

The orchestrator application is responsible for analyzing messages coming from the data plane MiddleModule nodes and for generating responses to these nodes. These messages are not generated with high frequency. They are standard TCP messages, which provide some predefined information in their payload portion. Therefore, a lightweight orchestrator application can accurately process these messages and generate the appropriate responses.

To provide the MiddleModule functionality to edge-switches, the modification, shown in ***Figure 5.11***, is suggested to be applied to all edge-switches. For packet monitoring purpose, an ASIC-based module can be used to mirror the header portions of the incoming packets through the processing block. The header mirror module should operate at high-speeds to minimize the system performance degradation. Packet processing algorithms are not that complex either and they do

not require high clock speeds. The processing block may receive different packets from different header mirror modules, simultaneously.

The power consumption of the processing block should be minimal. Therefore, a parallel processor unit, such as an FPGA, is proposed as a parallel processing block for each edge-switch. The memory unit of the processing block can be built by using either RAM and/or TCAM. Small memory blocks would be sufficient for this purpose since the size of the tables kept for each connected host is limited, and the required number of parameters of detection algorithms is small. The attack prevention functionality can be implemented by using OpenFlow commands. During the evaluations applied in the Mininet, we used this technique. The Packet Processing Block can modify the flow table of the Forwarding Circuit by sending OpenFlow commands, to implement the attack prevention rules.

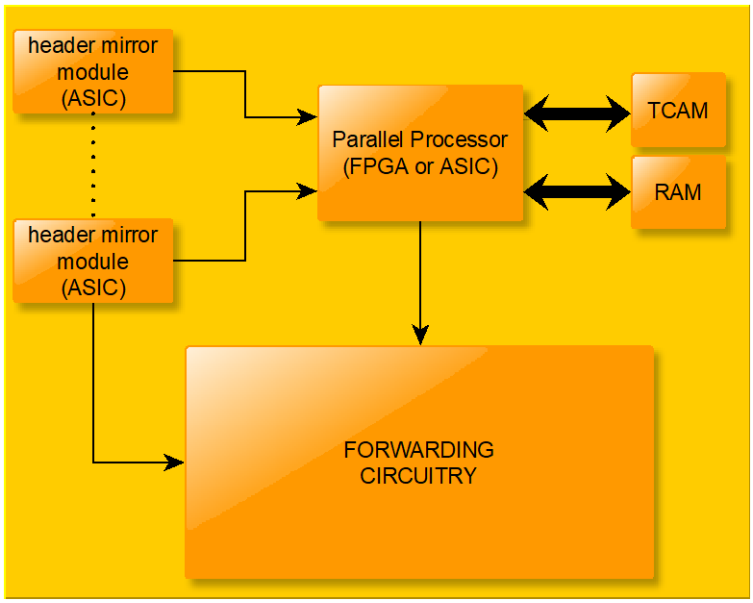


Figure 5.11: Data Plane Device MiddleModule Implementation Suggestion

5.3.5 Compromise-Ability Discussion

The MiddleModule system uses data plane monitoring, detection and prevention nodes, distributed across the network and working in a stand-alone fashion.

Therefore, the MiddleModule system does not have any single point of failure problem unlike the controller-based defense systems. If a defense node malfunctions for any reason, other nodes can continue their operations without any problem. Furthermore, as was stated and tested earlier, the MiddleModule system can operate with different protocols and traffic rates; hence fluctuating traffic or a high packet generation rate does not cause any failure of the MiddleModule system. In addition, since the defense nodes send *malicious information messages* and also, they can send periodic status messages, if desired, to the controller; the controller can analyze the decisions of each data plane node. If any defense node is compromised by a malicious user, and the node starts malfunctioning, the controller can detect that malfunctioning node. The controller can then fix the problem or can deactivate the node entirely by changing the detection algorithm parameters or by defining whitelist and blacklists. The configuration channels of the data plane nodes, the channel between the data plane nodes and the controller, can become the weak spot of the nodes. However, in MiddleModule system, edge-switches have dedicated configuration channel ports, which are different from the regular ingress ports. If required, the communication on that configuration channel can be secured with encryption or using any other method.

5.4 Comparison and Further Discussion

In this section, MiddleModule system is compared with similar studies found in the literature using performance metrics mentioned earlier. In this thesis the following data plane-based DDoS defense systems are considered during comparison; namely, AVANT-GUARD [38], CIDS [41], StateSec [4] and SDNScore [2].

Although all these studies and the MiddleModule system are proposed against DDoS attacks; the targeted attack types in the above studies are slightly different than each other. AVANT-GUARD system considers only SYN Flood DDoS attack type, CIDS system targets only SYN Flood attacks and Flooding attacks (ICMP Flooding and UDP Flooding), while the StateSec system is tested against only the Flooding attacks. The SDNScore concentrates on SYN Flood attack and Flooding attacks. On the other hand, the MiddleModule system is designed and tested against IP Spoofing

attack, SYN Flood attack, Flooding attack with and without spoofing, Reflection attack and Broadcast Amplification attacks. Therefore, the MiddleModule system covers a larger Network/Transport-Level DDoS attack set.

The defense strengths of the studies are compared with our proposed method. For that purpose, we compare them under both ideal and non-ideal conditions. Ideal condition means that the attacks have well-defined packet attributes (packet lengths, protocols, etc.) and they are generated at normal-speed.

Since the data plane-based defensive approach is an effective method against Network-Level DDoS attacks, all the compared methods and also the MiddleModule system have remarkable defense strength values under ideal conditions, as shown in ***Table 5.15***.

Table 5.15: Defense Strength Comparison under Ideal Conditions

	SYN Flood attack		***ICMP or UDP Flooding attack***	
	Sensitivity	***False Positive Rate***	***Sensitivity***	***False Positive Rate***
CIDS	94.8%	2.3%	96.3 %	3.4 %
StateSec	-	-	~ 100 %	~ 0.0 %
SDNScore	100 %	0.02%	100 %	1.0 %
MiddleModule	99.3 %	0.0 %	99.3 %	0.0 %

Since there are no numeric defense strength values provided for AVANT-GUARD system, we cannot use it in defense strength comparison. As shown in ***Table 5.15***, the MiddleModule system provides similar defense strength performance against these attacks when compared with other studies.

However, under non-ideal conditions, the defense strength performance of the MiddleModule system is noticeably higher than other studies. As shown in ***Table 5.16***, when generation rate of malicious traffic is low, the sensitivity results change since the attack intensity decreases. The StateSec, CIDS and MiddleModule systems are tested under such conditions; the StateSec is tested with *1pckt/10ms*; the CIDS is

tested with a value between *1pckt/10ms and 1 pckt/2ms*; the MiddleModule is tested with *1pckt/10ms* packet generation rates. ***Table 5.16*** shows that while the detection rate of the StateSec and the CIDS systems drops to approximately 80 %, the detection rate of the MiddleModule system stays above 96 %, which is close to results collected under ideal conditions. SDNScore is not tested with low generation speeds but it is tested using attacks with both well-defined and random packet attributes (protocol type, destination port number, packet byte length, etc.). Attacks with random attributes are called as *generic attacks* at SDNScore study. When the defense system is tested by using *generic attacks* at nominal packet generation rates; the detection rate performance of SDNScore decreases significantly, to nearly 75.0%. On the other hand, the MiddleModule detection algorithms analyze the volume and variation of packet properties, regardless of the values of these properties. Therefore, when a malicious packet is generated having packet attributes with either well-known values or generic values, the same performance results are obtained. During the emulation of the MiddleModule system, malicious packets are generated by using real traffic and DDoS packet attributes are generated randomly. Hence, as shown in ***Table 5.16***, the detection rate of the MiddleModule system with generic malicious packet attributes is 99.3%.

Table 5.16: Defense Strength Comparison under Non-Ideal Conditions

	Flooding attack		***Flooding attack***	
	Sensitivity	***False Positive Rate***	***Sensitivity***	***False Positive Rate***
CIDS	~ 79.0 %	~ 5.0 %	-	-
StateSec	~ 80.0 %	-	-	-
SDNScore	-	-	75.0 %	1.0 %
MiddleModule	96.4 %	0.0 %	99.3 %	0.0 %
	with Low data generation rate		***with not well-known attack pattern***	

MiddleModule system is observed to perform remarkably well under ideal and non-ideal conditions, while the defense strength of others decreases notably under non-ideal conditions.

The system performance degradation is the other important aspect especially for data plane-based defense systems, because in such system, all packets in the network experience the performance degradation caused by the defense system. A defense system should detect malicious activities accurately, but at the same time, it should cause minimal delay on the nominal user's connectivity. The delays on packet retrieval time of a nominal user, caused by the MiddleModule system and the AVANT-GUARD system, are compared in ***Table 5.17***.

Table 5.17: Packet Retrieval Time Delay Comparison

	AVANT-GUARD	***MiddleModule***
Packet retrieval time delay	1.86 %	0.2 %

The packet retrieval times of a nominal user with and without the defense system are recorded; and the packet retrieval delay, caused by the defense system is computed. When this test is applied for AVANT-GUARD system, the packet retrieval time is 1.86% longer. With MiddleModule system, packet retrieval time of a nominal user is 0.2% longer. Hence, MiddleModule causes nearly no delay on the packet retrieval time of a nominal user, which is remarkable when compared with the AVANT-GUARD system. Although this metric is a critical one, the other defense systems do not provide such test results. To evaluate system performance degradation of those systems, they provide only the communication overhead caused by the defense system. The communication overhead values of the other systems are less than 1%; on the other hand, the communication overhead of the MiddleModule system is less than 0.01 %, which is also remarkable when compared with the other studies. In fact, since the defense nodes in the MiddleModule system operates in a stand-alone fashion with lightweight detection algorithms, such system performance degradation results are expected for the MiddleModule system.

One of the most significant properties of Network/Transport-Level DDoS attacks is that they are generated at higher rates than normal conditions. Generally, if a host generates Network/Transport-Level DDoS traffic into the network, let's say a Flooding Traffic; it generates the traffic with the highest possible generation rate.

Therefore, it is critical to evaluate the proposed defense systems using high traffic generation rates. If a defense system is tested only under low packet rates than today's needs than the defense strength for that system could become meaningless under real networking conditions. ***Table 5.18*** is built by using the minimum and maximum packet generation rates, reported in the proposed defense systems. Some packet generation rates, used in ***Table 5.18***, are not directly provided in the corresponding study, such as in CIDS, but these values are inferred from the provided information in those studies. In addition, in SDNScore study, no packet generation rate information is provided.

Table 5.18: The Traffic Generation Rates Comparison

	Minimum packet generation rate	***Maximum packet generation rate***
AVANT-GUARD	*1 pckt / 10 ms (100 pckt / sec)*	*1 pckt / 1 ms (1000 pckt / sec)*
CIDS	*1 pckt / 1 ms (1000 pckt / sec)*	*1 pckt / 200 µs (5000 pckt / sec)*
StateSec	*1 pckt / 10 ms (100 pckt / sec)*	*1 pckt / 1200 µs (833 pckt / sec)*
MiddleModule	*1 pckt / 10 ms (100 pckt / sec)*	*1 pckt / 2 µs (500000 pckt / sec)*

As shown in ***Table 5.18***, the minimum packet generation rates are similar to each other, but the maximum rates are different. They change between 0.8 kpps and 5 kpps for the other studies while for the MiddleModule system, the maximum packet generation rate used during the evaluation is hundred times higher than the other studies (500 kpps). If the average packet length is assumed to be 250 bytes, then 5 kpps corresponds only 10Mbps while 500 kpps corresponds 1 Gpbs, which is more realistic in today's advanced networking technologies. Therefore, MiddleModule system is designed to operate at higher packet generation rates and also tested under more realistic conditions.

Some other aspects are also considered during comparison of the MiddleModule system and the other studies. Although providing a high-performance defense system is important, it is also essential to provide a system that meets critical characteristics of DDoS attacks, SDN and prevention systems.

The AVANT-GUARD system detects only SYN Flooding attacks with spoofed IP address. However, as was explained earlier, several other DDoS attack types could be dangerous for SDN. To prevent other Flooding attacks, an additional detection system should be implemented in the network. The CIDS system provides only the information whether DDoS exists or not; but it cannot provide source hosts and victim hosts of the attack. However, without that information, an attack cannot be prevented effectively. The StateSec system cannot operate properly if the DDoS traffic in the network is much stronger than the nominal traffic. The detection performance of the StateSec system decreases dramatically under that condition. However, in general, during a DDoS attack, DDoS traffic becomes much stronger. In addition, it cannot detect the source of malicious activities if the number of attackers exceeds a certain value. The SDNScore system cannot operate accurately if the malicious packets do not have well-defined attributes; however, malicious packets can have random attributes, too. In addition, it suggests a considerably complex detection mechanism, but it does not provide any test results about the connectivity delays caused by that detection mechanism system. In addition, it does not provide the detection performance variation under higher packet generation rates.

These aspects are critical for DDoS defense systems, and a comprehensive defense system should perform well when these aspects are considered.

When the MiddleModule system is compared with these studies in terms of these aspects, the MiddleModule system has remarkable features. The MiddleModule system detects several Network/Transport-Level attack types. It provides both source and victim hosts of a malicious activity. The MiddleModule can operate if the DDoS traffic is much stronger than the legitimate traffic, without any noticeable loss of performance. It accurately detects attacks with well-defined patterns or random pattern. Finally, the MiddleModule system uses lightweight detection algorithms; hence the MiddleModule does not cause any noticeable system performance degradation and it can detect malicious activities accurately even if they are generated using high rates.

Although the MiddleModule system has some points that require attention, such as setting detection algorithm parameters, the MiddleModule system is superior to other data plane-based defense systems when all aspects are considered.

Finally, the MiddleModule system is compared with one promising control plane-based defense system and one hybrid system. For this purpose, [24] and [33] are considered. In [24], Kokila *et al.* proposes using Support Vector Machine (SVM) for attack detection purposes. In this method, SVM-based detection algorithm runs on the controller. In addition, in that study, SVM is compared with several other classifiers. In [33], Rengaraju *et al.* proposes a hybrid system, where OpenFlow switches run simple anomaly detection algorithms. If switch detects any anomalous traffic or receives packets from unknown sources, it directs those packets to the controller. The controller applies IDS algorithms on those packets to detect any signature mismatch.

First, we compare our MiddleModule system with these systems in terms of defense strength. Rengaraju provides detection rate values for a test network including one attacker (ICMP and SYN flooding), one legitimate user and one victim server. The malicious traffic is injected to the network at 10 kpps injection rate. On the other hand, Kokila tests its SVM-based centralized defense system with different network topologies with legitimate users, malicious users and a single victim server. During the evaluation of this study, after setting the detection algorithm parameters to their optimum values, the defense system is trained for 120 seconds. After that, accuracy of the defense system is measured to be 95.1%. Then, in this study, accuracy result for SVM is compared with several detection methods including RBF, Naïve Bayes, Bagging, etc., and SVM method is measured to have better defense strength values.

When the MiddleModule system is compared with these systems, as shown in ***Table 5.19***, MiddleModule system provides superior defense strength performance than the other studies. The detection rate of Rengaraju is below 90% and the detection rate of Kokila is 95.1%. On the other hand, MiddleModule system provides a detection rate above 99%.

Table 5.19: Defense Strength Comparison with Control Plane-Based System and Hybrid System

	Rengaraju	***MiddleModule***	***Kokila***	***MiddleModule***
Flooding Attack	89.4 %	99.3 %	95.1 %	99.6 %
	Detection Rate		***Accuracy***	

It should also be noted that, the main advantage of the MiddleModule system is, it avoids single-point of failure problem. The other systems analyze packets at control plane level, and as the attack size increases, the workload of the controller also increases. After a point, controller resources would exhaust, as was explained earlier. Although this is a critical issue, these centralized methods do not provide any test results regarding this problem; they analyze only defense strength performance.

All in all, the MiddleModule system provides superior results when compared with both data plane-based methods and centralized methods regarding all aspects.

CHAPTER 6

CONCLUSION AND FUTURE WORK

Network/Transport-Level DDoS is a challenging issue for SDN architecture, and these attacks cause devastating problems for SDN, which may lead to failure of an entire SDN network. There is no generally accepted solution against Network/Transport-Level DDoS attacks for SDN networks in the literature; and in fact, there are many unresolved problems in this area. The existing methods suggest to deploy the defense systems either in control plane or in data plane. Although some of the control plane-based systems provide promising defense strength values, they increase the computational load of the controller. In addition, they require message exchanges between the switches and the controller for accurate decisions, which congests the communication channel even more under an organized DDoS attack. Furthermore, they have single point of failure problems and they can be neutralized under a sophisticated DDoS attack. However, data plane-based approaches provide more robust defense systems. There are a few data plane-based DDoS defense mechanisms provided in the literature; and as was mentioned in the Chapter V, they have certain drawbacks regarding the evaluation metrics.

In this thesis, the MiddleModule DDoS detection and prevention system is proposed to defend SDN networks against Network/Transport-Level DDoS attacks. The MiddleModule is a data plane-based defense system, deploying the monitoring, detection and prevention capabilities in data plane nodes. Before designing this system, the requirements that a data plane-based system should satisfy are analyzed and explained. The MiddleModule system is designed by considering these requirements. In this system, edge-switches of the network are modified so that they monitor some statistics of the incoming packets and operate simple threshold-based detection functions by using these statistics and makes flow-based decisions. In the

detection functions, the variations of some packet attributes, depending on the DDoS attack types are analyzed. The controller is also informed according to detection decisions and the appropriate prevention method is applied by on the associated edge-switch. The MiddleModule system is tested in different evaluation platforms to obtain comprehensive and accurate performance results. The evaluation results suggest that the MiddleModule system operates as required under different test scenarios. The MiddleModule system provides negligible system performance degradation and remarkable defense strength performances against Network/Transport-Level DDoS attacks, under both ideal and non-ideal conditions. Test results also suggest that the MiddleModule system can operate accurately when malicious hosts generate packets at both high and low generation rates. The test results are compared with the four more recent and promising data plane-based defense systems. The comparison results suggest that our proposal, the MiddleModule system, is superior to the existing studies in the literature, regarding most of the evaluation metrics, and comparable regarding many other aspects.

In this thesis, both OMNET++ and Mininet platforms are used in evaluations. The simulation and emulation environments can be compared while analyzing the evaluation results. Both OMNET++ and Mininet based tests have provided similar results especially during the functional analysis phase. OMNET++ provides more flexible and reproducible tests; hence, it may be possible to suggest using this simulation tool in system design in general. On the other hand, Mininet provides more detailed performance analysis results in a more realistic testing environment; hence, we suggest using this emulation tool during proof-of-work phases.

When the drawbacks and possible future works are analyzed, we notice three issues. First, in the current MiddleModule system, the values of detection algorithm parameters are assigned manually. If they are not assigned appropriately, accuracy of the MiddleModule system decreases. Therefore, these variables, namely *windowSize*, *saturationThreshold* and *maliciousThreshold*, should be assigned automatically as future work to avoid such problems. For that purpose, MiddleModule Orchestrator module, which runs on the controller, can generate appropriate values for these parameters by monitoring detection decisions of the switches and by considering the

network topology. Then, these values can be distributed to each data plane detection node. In addition, malicious list updating technique applied by each edge-switch can be modified so that any entry is stored to that list after receiving a confirmation from controller. Second, in the scope of this thesis, the Network/Transport-Level DDoS attacks are considered only. The MiddleModule system can be improved to defend an SDN network against more sophisticated attacks along with Application-Level DDoS attacks as future work. Sophisticated but volumetric attacks can be searched at data plane devices by using more collaborative approach with more detailed detection algorithms. Although some other techniques can be used for this purpose, using Network Function Virtualization (NFV) can provide certain advantages. The orchestrator of NFV can be used to provide collaboration between data plane detection nodes. It can easily analyze the performance of each data plane detection node, activate/deactivate them automatically, modify their parameters or update their detection lists or algorithms quickly. In addition, sophisticated detection algorithms can be generated as virtual functions by a centralized and powerful device and the required algorithms can be deployed into the required data plane devices in coordination by using the orchestrator of NFV. Furthermore, to cover sophisticated but non-volumetric attacks, a centralized detection method can be operated in agreement with the data plane-based approach suggested in this work. Finally, the MiddleModule system benefits from distributed detection nodes. In fact, to get the best performance, the detection nodes should be distributed into the network with maximum distribution, so that they should be placed close to hosts. This increases the implementation complexity of the system. As future work, using a collaborative approach could be useful to decrease such implementation complexity.

REFERENCES

[1] 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," *IEEE Commun. Surv. Tutorials*, vol. 18, no. 1, pp. 602–622, 2016.

[2] K. Kalkan, G. Gur, and F. Alagoz, "SDNScore: A statistical defense mechanism against DDoS attacks in SDN environment," *Proc. - IEEE Symp. Comput. Commun.*, pp. 669–675, 2017.

[3] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDOS) flooding attacks," *IEEE Commun. Surv. Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[4] J. Boite, P. A. Nardin, F. Rebecchi, M. Bouet, and V. Conan, "Statesec: Stateful monitoring for DDoS protection in software defined networks," *2017 IEEE Conf. Netw. Softwarization Softwarization Sustain. a Hyper-Connected World en Route to 5G, NetSoft 2017*, 2017.

[5] F. Ieee *et al.*, "Software-Defined Networking : A Comprehensive Survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[6] "Open Datapath Standardized Switch Protocol in Software Defined Network (SDN)." [Online]. Available: https://www.opennetworking.org/projects/open-datapath/. [Accessed: 01-Jan-2018].

[7] "Software-Defined Networking (SDN) Definition - Open Networking Foundation." [Online]. Available: https://www.opennetworking.org/sdn-definition/. [Accessed: 01-Jan-2018].

[8] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks Nick," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69, 2008.

[9] S. M. Mousavi and M. St-Hilaire, "Early detection of DDoS attacks against SDN controllers," *2015 Int. Conf. Comput. Netw. Commun. ICNC 2015*, pp. 77–81, 2015.

[10] M. Conti, A. Gangwal, and M. S. Gaur, "A comprehensive and effective mechanism for DDoS detection in SDN," *2017 IEEE 13th Int. Conf. Wirel. Mob. Comput. Netw. Commun.*, pp. 1–8, 2017.

[11] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "Sdn Security: A Survey," in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, 2013, pp. 1–

7.

[12] X. Huang, X. Du, and B. Song, “An effective DDoS defense scheme for SDN,” *IEEE Int. Conf. Commun.*, 2017.

[13] S. Jantila and K. Chaipah, “A Security Analysis of a Hybrid Mechanism to Defend DDoS Attacks in SDN,” *Procedia Comput. Sci.*, vol. 86, no. March, pp. 437–440, 2016.

[14] N. N. Dao, J. Park, M. Park, and S. Cho, “A feasible method to combat against DDoS attack in SDN network,” *Int. Conf. Inf. Netw.*, vol. 2015–Janua, pp. 309–311, 2015.

[15] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin, and Z. Zhang, “Enabling security functions with SDN: A feasibility study,” *Comput. Networks*, vol. 85, no. 2015, pp. 19–35, 2015.

[16] D. Hyun, J. Kim, and D. Hong, “SDN-based Network Security Functions for Effective DDoS Attack Mitigation,” pp. 834–839, 2017.

[17] P. Van Trung, T. T. Huong, D. Van Tuyen, D. M. Duc, N. H. Thanh, and A. Marshall, “A multi-criteria-based DDoS-attack prevention solution using software defined networking,” *2015 Int. Conf. Adv. Technol. Commun.*, pp. 308–313, 2015.

[18] Y. Cui *et al.*, “SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks,” *J. Netw. Comput. Appl.*, vol. 68, pp. 65–79, 2016.

[19] A. Hussein, I. H. Elhajj, A. Chehab, and A. Kayssi, “SDN security plane: An architecture for resilient security services,” *Proc. - 2016 IEEE Int. Conf. Cloud Eng. Work. IC2EW 2016*, pp. 54–59, 2016.

[20] S. Luo, J. Wu, J. Li, and B. Pei, “A Defense Mechanism for Distributed Denial of Service Attack in Software-Defined Networks,” *2015 Ninth Int. Conf. Front. Comput. Sci. Technol.*, pp. 325–329, 2015.

[21] Nishat-I-Mowla, I. Doh, and K. Chae, “Multi-defense mechanism against DDoS in SDN based CDNi,” *Proc. - 2014 8th Int. Conf. Innov. Mob. Internet Serv. Ubiquitous Comput. IMIS 2014*, pp. 447–451, 2014.

[22] J. Ashraf and S. Latif, “Handling intrusion and DDoS attacks in Software Defined Networks using machine learning techniques,” *2014 Natl. Softw. Eng. Conf.*, pp. 55–60, 2014.

[23] T. V Phan, N. K. Bao, and M. Park, “A Novel Hybrid Flow-based Handler with DDoS Attacks in Software-Defined Networking,” no. November, pp. 350–357, 2016.

[24] K. RT, S. T. Selvi, and K. Govindarajan, “DDoS Detection and Analysis in SDN-Based Environment Using Support Vector Machine Classifier,” *6th Int. Conf. Adv. Comput. (ICoAC 2014)*, pp. 205–210, 2014.

[25] L. Barki, A. Shidling, N. Meti, D. G. Narayan, and M. M. Mulla, “Detection Of Distributed Denial Of Service Attacks In Software Defined Networks,” *2016 Int. Conf. Adv. Comput. Commun. Informatics*, pp. 2576–2581, 2016.

[26] R. Braga, E. Mota, and A. Passito, “Lightweight DDoS flooding attack detection using NOX/OpenFlow,” *Proc. - Conf. Local Comput. Networks, LCN*, pp. 408–415, 2010.

[27] Y. Xu and Y. Liu, “DDoS attack detection under SDN context,” *Proc. - IEEE INFOCOM*, vol. 2016–July, 2016.

[28] N. I. G. Dharma, M. F. Muthohar, J. D. A. Prayuda, K. Priagung, and D. Choi, “Time-based DDoS detection and mitigation for SDN controller,” *17th Asia-Pacific Netw. Oper. Manag. Symp. Manag. a Very Connect. World, APNOMS 2015*, pp. 550–553, 2015.

[29] P. Dong, X. Du, H. Zhang, and T. Xu, “A detection method for a novel DDoS attack against SDN controllers by vast new low-traffic flows,” *2016 IEEE Int. Conf. Commun. ICC 2016*, 2016.

[30] J. Kim, M. Daghmehchi Firoozjaei, J. P. Jeong, H. Kim, and J. S. Park, “SDN-based security services using interface to network security functions,” *Int. Conf. ICT Converg. 2015 Innov. Towar. IoT, 5G, Smart Media Era, ICTC 2015*, pp. 526–529, 2015.

[31] B. Yuan, D. Zou, S. Yu, H. Jin, W. Qiang, and J. Shen, “Defending against Flow Table Overloading Attack in Software-Defined Networks,” *IEEE Trans. Serv. Comput.*, vol. 1374, no. c, pp. 1–1, 2016.

[32] S. Yang, Y. Kim, H. Kim, S. Yang, and S. Lim, “Controller scheduling for continued SDN operation under DDoS attacks,” *Electron. Lett.*, vol. 51, no. 16, pp. 1259–1261, 2015.

[33] P. Rengaraju, R. R. V, and C.-H. Lung, “Detection and Prevention of DoS attacks in Software-Defined Cloud Networks,” *Dependable Secur. Comput. 2017 IEEE Conf.*, pp. 217–223, 2017.

[34] T. Chin, X. Mountrouidou, X. Li, and K. Xiong, “An SDN-supported collaborative approach for DDoS flooding detection and containment,” *Proc. - IEEE Mil. Commun. Conf. MILCOM*, vol. 2015–Decem, pp. 659–664, 2015.

[35] C. C. Machado, L. Z. Granville, and A. Schaeffer-Filho, “ANSwer: Combining NFV and SDN features for network resilience strategies,” *Proc. - IEEE Symp. Comput. Commun.*, vol. 2016–Augus, pp. 391–396, 2016.

[36] “OpenFlow Switch Specification,” 2014.

[37] R. Wang, Z. Jia, and L. Ju, “An entropy-based distributed DDoS detection mechanism in software-defined networking,” *Proc. - 14th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. Trust. 2015*, vol. 1, pp. 310–317, 2015.

[38] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, “AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-Defined Networks,” *ACM SIGSAC Conf. Comput. Commun. Secur. (CCS 2013)*, pp. 413–424, 2013.

[39] K. Y. Chen, A. R. Junuthula, I. K. Siddhrau, Y. Xu, and H. J. Chao, “SDNShield: Towards more comprehensive defense against DDoS attacks on SDN control plane,” *2016 IEEE Conf. Commun. Netw. Secur. CNS 2016*, pp. 28–36, 2017.

[40] Y. Kim and H. J. Chao, “PacketScore: Statistics-based Overload Control against Distributed Denial-of-Service Attacks,” pp. 2594–2604, 2004.

[41] X. Chen and S. Yu, “A collaborative intrusion detection system against DDoS for SDN,” *IEICE Trans. Inf. Syst.*, vol. E99D, no. 9, pp. 2395–2399, 2016.

[42] X. F. Chen and S. Z. Yu, “CIPA: A collaborative intrusion prevention architecture for programmable network and SDN,” *Comput. Secur.*, vol. 58, pp. 1–19, 2016.

[43] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, “Reproducible network experiments using container-based emulation,” *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol. - Conex. ’12*, p. 253, 2012.

[44] H. Ringberg, M. Roughan, and J. Rexford, “The need for simulation in evaluating anomaly detectors,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 1, p. 55, 2008.

[45] “OMNeT++ Discrete Event Simulator - Home.” [Online]. Available: https://www.omnetpp.org/. [Accessed: 24-Feb-2018].

[46] “INET Framework - INET Framework.” [Online]. Available: https://inet.omnetpp.org/. [Accessed: 05-Feb-2018].

[47] D. Klein and M. Jarschel, “An OpenFlow Extension for the OMNeT++ INET Framework,” *Proc. Sixth Int. Conf. Simul. Tools Tech.*, 2013.

[48] “Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet.” [Online]. Available: http://mininet.org/. [Accessed: 05-Feb-2018].

[49] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6.

[50] “KDD Cup 1999 Data.” [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. [Accessed: 05-Feb-2018].

[51] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, “MAWILab : Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking.”

[52] “Scapy.” [Online]. Available: http://www.secdev.org/projects/scapy/. [Accessed: 06-Feb-2018].

[53] “packeth.” [Online]. Available: http://packeth.sourceforge.net/packeth/Home.html. [Accessed: 06-Feb-2018].

[54] “Download/Get Started with Mininet - Mininet.” [Online]. Available: http://mininet.org/download/. [Accessed: 02-Feb-2018].

[55] A. A. Ghorbani, W. Lu, and M. Tavallaee, “Network Intrusion Detection And Prevention Middlebox Management In SDN,” vol. 47, 2010.