

**SAKARYA UNIVERSITY  
INSTITUTE OF SCIENCE AND TECHNOLOGY**

**ANOMALY DETECTION IN  
SOFTWARE-DEFINED NETWORKING  
USING MACHINE LEARNING**

**M.Sc. THESIS**

**Soumaine BOUBA MAHAMAT**

**Department : COMPUTER AND INFORMATION  
ENGINEERING**

**Supervisor : Prof. Dr. Celal ÇEKEN**

**June 2018**

SAKARYA UNIVERSITY  
INSTITUTE OF SCIENCE AND TECHNOLOGY

**ANOMALY DETECTION IN  
SOFTWARE-DEFINED NETWORKING  
USING MACHINE LEARNING**

**M.Sc. THESIS**

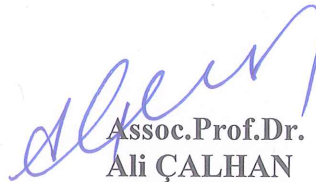
**Soumaine BOUBA MAHAMAT**

**Department : COMPUTER AND INFORMATION  
ENGINEERING**

**This thesis has been accepted unanimously / with majority of votes by the  
examination committee on 28.06.2018.**



**Prof.Dr.  
Celal ÇEKEN  
Head of Jury**



**Assoc.Prof.Dr.  
Ali ÇALHAN  
Jury Member**



**Assist.Prof.Dr.  
Seçkin ARI  
Jury Member**

## **DECLARATION**

I declare that all the data in this thesis was obtained by myself in academic rules, all visual and written information and results were presented in accordance with academic and ethical rules. There is no distortion in the presented data. In case of utilizing other people's works they were refereed properly according to scientific norms. The data presented in this thesis has not been used in any other thesis in this university or in any other university.

Soumaine BOUBA MAHAMAT  
28.6.2018

## ACKNOWLEDGEMENT

Firstly, I would like to thank ALLAH for keeping me healthy and helping me complete my thesis. Without his help, I wouldn't be where I am now.

With deep appreciation and respect, I would like to give credit to my research and thesis supervisor Prof. Dr. Celal ÇEKEN for his great support since I started my research and for allowing me to be part of his research team at the IoT Laboratory in the Department of Computer and Information Engineering.

My thanks are also to YTB (*Yurtdışı Türkler ve Akraba Topluluklar Başkanlığı*), the institution which financed my studies and stay for three years in Turkey. To my family members, my friends in Sakarya, and my collaborators in the IoT Lab especially Ahmed DIRIE, thank you for everything.

## TABLE OF CONTENTS

ACKNOWLEDGMENT.....	i
TABLE OF CONTENTS.....	ii
LIST OF SYMBOLS AND ABBREVIATIONS.....	iv
LIST OF FIGURES.. ..	v
LIST OF TABLES .....	vi
SUMMARY.....	vii
ÖZET .....	viii
CHAPTER 1.	
INTRODUCTION.....	1
1.1. Problem statement.....	5
1.2. Thesis goals .....	5
1.3. Thesis structure .....	5
CHAPTER 2.	
BACKGROUND AND LITERATURE REVIEW.....	6
2.1. History of SDN.....	6
2.2. SDN security overview.....	9
2.3. Network Anomaly Detection: A Machine Learning Perspective .....	11
2.4. Literature Review .....	11
CHAPTER 3.	
METHODOLOGY.....	13
3.1. Intrusion detection.....	13
3.2. Controller of choice: POX.....	14
3.3. OpenFlow Protocol.....	15
3.3.1. OpenFlow switch.....	15

3.3.2. Flow table.....	16
3.3.3. Flow stats.....	16
3.4. Data Collection.....	17
3.5. Training the machine learning algorithm.....	17
3.5.1. Python Scikit-learn.....	18
3.5.2. Decision tree algorithm.....	18
3.5.3. K Nearest Neighbors (KNN).....	19
3.5.4. Confusion Matrix.....	19
3.5.5. Accuracy.....	20
3.6. DDoS flood attack: detection and mitigation.....	21
3.7. Limitations of the project.....	21
CHAPTER 4.	
EXPERIMENTAL IMPLEMENTATION.....	22
4.1. Tools used for the testbed.....	22
4.1.1. Mininet.....	22
4.1.2. POX Controller.....	22
4.1.3. Our Machine Learning Algorithm.....	23
4.2. Environment .....	25
4.2.1. Topology.....	25
4.2.2. Attack Scenario.....	27
4.3. Results.....	28
CHAPTER 5.	
CONCLUSION ANF FUTURE WORK.....	29
REFERENCES.....	30
RESUME.....	34

## **LIST OF SYMBOLS AND ABBREVIATIONS**

API	: Application Programming Interface
BGP	: Border Gateway Protocol
CART	: Classification and Regression Trees
CE	: Control Elements
CLI	: Command Line Interface
DC	: Domain Controller
DDoS	: Distributed Denial of Service
DoS	: Denial of Service
DT	: Decision Trees
FE	: Forward Elements
ICMP	: Internet Control Message Protocol
ID3	: Iterative Dichotomiser 3
IETF	: Internet Engineering Task Force
IT	: Information Technology
LAN	: Local Area Network
NCP	: Network Control Point
NFV	: Network Function Virtualization
ONF	: Open Networking Foundation
RCP	: Routing Control Point
SDN	: Software Defined Networking
SYN	: Synchronize
TCP	: Transmission Control Protocol
VINI	: Virtual Network Infrastructure
VLAN	: Virtual Local Area Network
VM	: Virtual Machine

## LIST OF FIGURES

Figure 1.1. SDN Architecture .....	1
Figure 1.2. Ping Demo using SDN Network .....	2
Figure 1.3. Poseidon Stack .....	4
Figure 2.1. Google’s OpenFlow WAN .....	9
Figure 2.2. Attacks in SDN .....	10
Figure 2.3. Taxonomy of Attack Related Tools .....	11
Figure 2.4. Detection Loop Operation .....	12
Figure 3.1. Intrusion Detection Systems: A Taxonomy .....	14
Figure 3.2. Main Components of an OpenFlow Switch .....	15
Figure 3.3. Structure of Match Information .....	16
Figure 3.4. Structure of Flow Information .....	16
Figure 3.5. Training the Machine Learning Algorithm.....	17
Figure 3.6. DDoS Attack Structure .....	21
Figure 4.1. Import Libraries .....	23
Figure 4.2. Dataset .....	23
Figure 4.3. Classification .....	24
Figure 4.4. Making Prediction .....	24
Figure 4.5. Evaluating the Model .....	25
Figure 4.6. Model Persistence .....	25
Figure 4.7. Launching the POX Controller and the IDS Module.....	25
Figure 4.8. Launching the Network Topology .....	26
Figure 4.9. Flow Stats .....	26
Figure 4.10. Topology Created in Mininet .....	27
Figure 4.11. Sequence Diagram .....	28
Figure 4.12. Ping Flood Attack Detection and Mitigation.....	28



## **LIST OF TABLES**

Table 3.1. Confusion Matrix (General) .....	19
Table 3.2. Confusion Matrix (Our Project).....	20

## **SUMMARY**

Keywords: Software-Defined Networking, Anomaly Detection, Machine Learning, Testbed

In recent years, the Software-Defined Networking (SDN) approach has emerged with the aims of making computer networks more flexible. Although the SDN application on Google's internal network demonstrates the usefulness of the Software-Defined Network approach and the promise of future technology, security is a vital concern that cannot be ignored. In the SDN architecture, the attacker can now attack the network from any of the three planes because the Data Plane is separated from the Control Plane. Machine learning algorithms are methods used to detect attacks and intrusions on computer networks and can also be used for SDN.

In this study, a new testbed has been implemented for anomaly detection in SDN. The testbed formed has several components such the POX controller, an IDS module implemented in POX, and a trained machine learning algorithm. The system examines flow statistics sent by OpenFlow switches and decides whether they are benign or malicious. In case a malicious flow has been detected, the controller will send a flow modification message, instructing the switch to block the attacking host from communicating in the network. To validate the testbed, a ping flood attack has been launched and malicious flows were detected and successfully blocked. Experimental results show that using machine learning, flow-based anomaly detection in SDN can be realized.

# YAZILIM TANIMLI AĞLARDA MAKİNE ÖĞRENİMİ İLE ANOMALİ TESPİTİ

## ÖZET

Anahtar kelimeler: Yazılım Tanımlı Ağ, Anomali Tespiti, Makine Öğrenimi, Test Düzenneđi.

Son yıllarda, bilgisayar ağlarını daha esnek bir hale getirmeyi amaçlayan Yazılım Tanımlı Ağ yaklaşımı ortaya çıkmıştır. Google'ın iç ağındaki Yazılım tanımlı ağ uygulaması, Yazılım Tanımlı Ağ (SDN) yaklaşımının kullanılabilirliğini ve gelecek vadeden bir teknoloji olacağını kanıtlanmasına rağmen güvenlik konusu göz ardı edilemeyecek hayati bir sorundur. SDN mimarisinde, Veri Düzlemini Kontrol Düzleminden ayrıldığı için saldırganlar artık üç düzlemde herhangi birinden ağa saldırabilirler. Makine öğrenimi algoritmaları, bilgisayar ağlarına yapılan saldırıları ve izinsiz girişleri tespit etmede kullanılan yöntemlerdir ve Yazılım Tanımlı Ağlar için de kullanılabilir.

Bu çalışmada Yazılım Tanımlı Ağlarda anomali tespiti için yeni bir test ortamı oluşturulmuştur. Oluşturulan test ortamının POX denetleyici, POX içerisinde geliştirilmiş bir IDS modülü ve eğitilmiş makine öğrenimi algoritması gibi çeşitli bileşenleri vardır. Sistem OpenFlow anahtarı tarafından gönderilen akış istatistiklerini inceleyerek bu akışa ait yararlı ya da zararlı kararını verir. Zararlı bir akış tespit edilmesi durumunda Yazılım Tanımlı Ağ denetleyici saldırganın ağ ile iletişiminin kesilmesi için anahtara akış düzenleme mesajı gönderir. Geliştirilen test ortamının doğrulanması için ping saldırısı uygulandı ve zararlı akışların sistem tarafından başarıyla tespit edilerek engellendiği görüldü. Elde edilen deneysel sonuçlar, makine öğrenimini kullanarak, Yazılım Tanımlı Ağlarda akış tabanlı anomali tespitinin gerçekleştirilebileceğini göstermektedir.

## CHAPTER 1. INTRODUCTION

The networking world is trying to catch up with the other branches of IT such as the Server Virtualization world. Networking folks are working hard to make networks more programmable. Computer networks are difficult to manage. They are made up of a lot of devices ranging from routers and switches to middleboxes. Unfortunately, nowadays, these devices are being managed on a box-by-box bases and network outages are mostly the direct results of human errors in configuring the networking devices [1], [2], [3], [4], [5]. Moreover, there is no platform that would allow us to have a single view of the network and can help us manage all of its components. These are among the reasons why Software Defined Networking (SDN) is happening.

SDN is a promising technology offering many possibilities. Vendors such as Cisco, HP, Juniper, etc. are greatly benefiting from the innovations that SDN is offering. In fact, Cisco has even revised its Certification track to include SDN related topics [6].

SDN is made up of three layers or planes: The Infrastructure Layer (Data Plane), The Control Layer (Control Plane), and the Application Layer (Application Plane) [7].

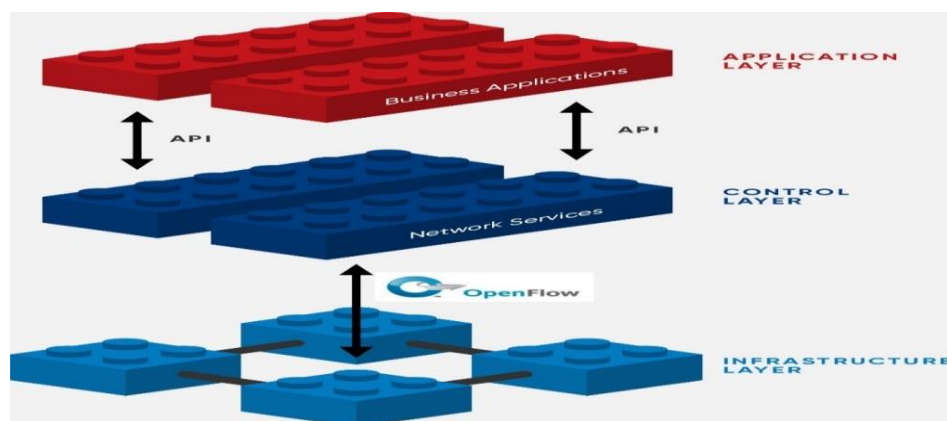


Figure 1.1. SDN Architecture [7]

With the decoupling of the Data Plane from the Control Plane, the Data Plane is now made up of devices that solely do forwarding. Through this decoupling, SDN promises that users would be able to buy switches from any vendor and use them in their networks.

The Control Plane is made of one or more controllers which control one or more switches in the Data Plane. The Control Plane allows us to have a general view of the network from a single point, something traditional networks can't generally do.

Using protocols such as OpenFlow, the Controller and switches exchange messages back and forth.

We would like to pinpoint a misconception here. Because the intelligence is removed from the switches in the Data Plane, a lot of people think that all packets have to go through the Controller in the Control Plane. The fact is that, only the first packet goes to the Controller. After that first ICMP request and reply, the flows are then cached into the switch's flow table for a limited time and therefore subsequent packets don't have to go through the Controller.

Figure 1.2. shows us the results of a host named h1 pinging another host named h2. As we can see, the first ping took 1.38ms however the subsequent pings took far less time. This is due to the fact that the first ping goes through the controller. The flow is then cached into the switch's flow table and subsequent pings are forwarded directly without the need to forward them to the controller.

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.38 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.205 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.109 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.264 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.127 ms
```

Figure 1.2. Ping Demo using an SDN Network

Through APIs (REST API for example), application developers can develop applications for the Application Plane. The Application Plane is believed to be what will make SDN worth it.

With the blessings of SDN comes a great challenge. Decoupling the Controlling Plane from Data Plane has its advantages; however, it opens up the doors to attacks. SDN networks can be attacked in many different ways. The control plane is considered as the brain of SDN and presents the risk of a single point of failure. If an attacker manages to compromise the controller, then the whole network is compromised. Despite this fact, it is mentioned that the benefits of a centralized control plane far outweigh the risk of single point of failure [8]. The single point of failure issue can be alleviated by ensuring high availability systems. To tackle this and other issues, security risks related to SDN could be mitigated through encryption, deep packet inspection, access control lists, etc [9].

SDN Security will be discussed in great details in Chapter 2: Background and Literature Review.

Machine learning techniques have long existed and used in different areas of computer science. With the emergence of SDN, researchers thought about implementing machine learning algorithms in this new technology. In a research paper titled knowledge-defined networking [10], the authors described experiments whereby data is gathered from hosts, processed using machine learning, and used the results to manage the SDN controller configuration and forwarding. A joint effort between Cisco and Ericsson resulted in a framework named MILA. Using time series data in OpenDaylight, they presented a demo on how to apply machine learning in SDN and generate meaningful data from it [11].

From a security perspective, a lot of effort has been done as well to demonstrate the usefulness of machine learning in SDN anomaly detection. The Poseidon project was a joint effort between CyberReboot and Lab41 [12]. Their goal was to investigate how machine learning could provide automation and security to SDN. Poseidon was

made up of three parts: A monitor module that interacts with the controller through a northbound API, a main module that updates the network through what it learned from the machine learning classifications, and a database to store models. Poseidon and similar efforts have proven that machine learning could be useful for intrusion detection in SDN.

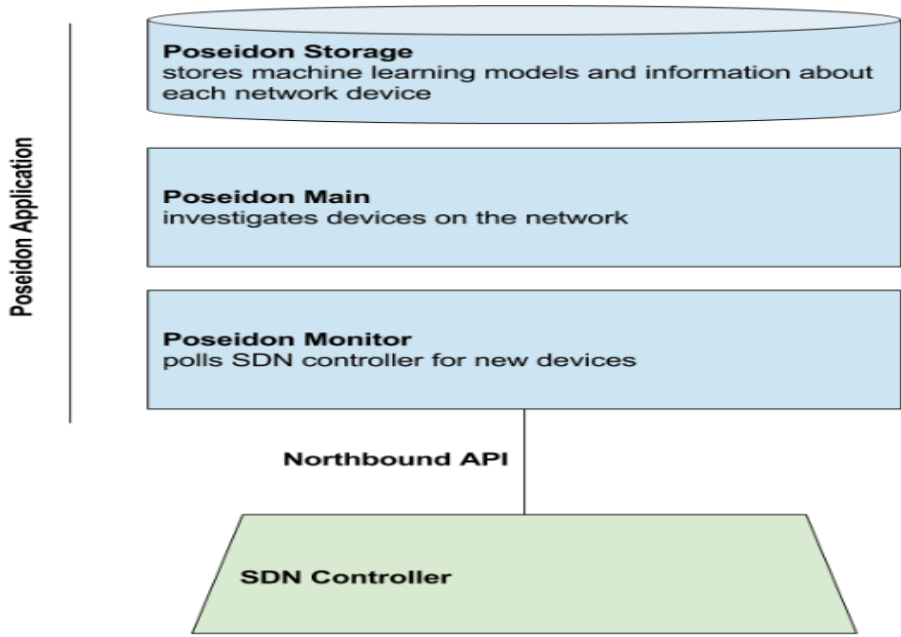


Figure 1.3. Poseidon Stack [12]

### **1.1. Problem Statement**

Networks have always been expensive, difficult to manage, and take longer to be implemented. This creates a lot of frustration in companies and governments. SDN has the goal of changing that. By decoupling the Control Plane from the Data Plane, a lot of opportunities are created. However, with the blessings of SDN comes a burden. As we open up the network, naturally attack vectors increase and security becomes a real concern. Networks are now being targeted from any of the three planes.

### **1.2. Thesis Goals**

Our goal is to create a flow-based anomaly detection system in SDN using machine learning algorithms. We will use an algorithm that would allow us to automatically detect and stop attacks.

### **1.3. Thesis Structure**

In Chapter 2, we will talk about the History of SDN and SDN Security in details. We will also discuss Machine Learning in SDN and do some Literature Review. Chapter 3 will discuss the methodology used in this Thesis. In Chapter 4, we will discuss the Implementation and Results of our research. Finally, we will conclude the Thesis in Chapter 5.



## **CHAPTER 2. BACKGROUND AND LITERATURE REVIEW**

A little history and literature review will be discussed in this chapter.

### **2.1. History of SDN**

Software Defined Networking (SDN) might seem at first as a completely new technology. However, it is a series of many efforts trying to make the network more flexible. Feamster *et. al.* [13] mentioned that some of the ideas of SDN date back to at least 20 years ago or even more. The authors also mentioned that some ideas in SDN root back to active networking, which had the goal of making the network programmable.

One of SDN's goals is Central Network Control. The origin of central network control can be traced back to at least the 1980s in the AT&T's Network Control Point (NCP) [14], [15]. They were geared towards Telephone Networks. The goal was to separate the Control Plane from the Data Plane. NCPs allowed AT&T to deploy services on demand and introduce new services rapidly. NCPs are still in use today.

In the 1990s, the concept of programmability in networks and network virtualization started emerging with various technologies such as Active Networks, Switchlets, and VINI.

As defined by [16], "An active network is a network in which the nodes are programmed to perform custom operations on the messages that pass through the node.". Middleboxes (Firewalls, proxies, etc.) are some examples of Active Networks. Active Networking was used by researchers to introduce the concept of programmable networks. It came out when the internet was taking off. The motivation behind active networks was to accelerate innovations as it took around 10 years from prototyping and testing to the deployment of new technologies. Somehow

SDN has the same motivation as active networking: Accelerate innovation and deployment in computer networks. Funding from agencies such as DARPA encouraged researchers to continue their research in Active Networking [17], [18]. However, network operators were frustrated with their inability to deploy new technologies in the network. Some of the reasons why active networking couldn't succeed was because of timing. There was no killer app or a clear application for active networks as there is for SDN (OpenFlow). Besides, unlike SDN, Active Networking put too much focus on the end user as a programmer instead of the network operator [19]. Moreover, hardware ASICs were also expensive. Some of the contributions of Active Networking include programmable functions and network virtualization. Notable projects in Active Networks include ANTS [20], SwitchWare [21], Smart Packets [22], Open Signaling [23], and Tempest (Switchlets) [24].

Let us now talk about network virtualization. As mentioned earlier, SDN has some of its roots from network virtualization as well. Network Virtualization is the ability to create logical networks that are separated from the underlying physical hardware infrastructure [25]. Network Virtualization allows us to create different logical topologies by using the same devices. Probably the most popular Network Virtualization Technology is Virtual LANs (VLANs), which are still in use today. Nowadays however, some of the most popular Network Virtualization technologies come from vendors such as Oracle, VMWare, Nicira, and Citrix. Other network virtualization technologies that existed earlier include Switchlets [24], and VINI.

VINI (Virtual Network Infrastructure) came into existence in 2006. The idea was to build a virtual network infrastructure that would allow experimenters and researchers to create their own virtual networks on top of the same underlying physical topology. VINI also had the goal of bridging the gap between lab or small-scale experiments and live deployment [26].

The History of Central Network Control was mostly in the context of Phone Networks and Circuit-Switched Networks. This discussion would be incomplete without introducing the History of Packet-Switched Networks.

In 2003, the FORCES protocol was standardized by the IETF [27], [28]. The standard defined protocols that would allow multiple Control Elements (CE) to control Forwarding Elements (FE). Again, we see the concepts of separating the Control Plane from the Data Plane, and Controllers controlling forwarding devices which exist in SDN here. The challenge for the protocol was that it required standardization, adoption, and deployment of new hardware (Which are the same problems of previous projects).

In 2004, the Routing Control Platform (RCP) came into existence [29]. RCP used the already existing BGP routing protocol to control the routing decisions made by BGP speaking routers. The main limitation of RCP was that it could handle only BGP speaking routers, while in reality a network operator might want to control a much wider range of routing protocols.

Ethane came out in 2007 [30] and offered a network architecture for the enterprise. In an Ethane network, all connectivity is controlled through a central server called the Domain Controller (DC). Clients joining the network for the first time have to connect to the DC first and be authenticated before getting access to other resources in the network. The drawback was that Ethane required custom switches that supported the protocol.

The ultimate goal was to find a solution that would work with existing protocols, yet it wouldn't require customizing the hardware. The answer was OpenFlow. OpenFlow [31] was developed by a group of engineers at Stanford University in 2008, with the first version being released in 2009 and updates in 2012 and 2015 [32]. Currently, OpenFlow is being managed by the Open Networking Foundation (ONF), a non-profit organization funded by companies such as Google, Facebook, HP, Microsoft, etc. [33]. OpenFlow has gotten so popular that it is being used in the live networks of some of these giant companies. In fact, in a presentation in 2012, Google's Urs Hoelzle described how Google was using the OpenFlow protocol to run its internal WAN. They first started to run OpenFlow enabled switches concurrently with

traditional switches, and then ended up removing all traditional network switches and their internal WAN is currently completely OpenFlow enabled [32].



Figure 2.1. Google's OpenFlow WAN [32]

## 2.2. SDN Security Overview

One of the hot topics in SDN is security. Many researchers and companies believe that Security will be one of the keys to SDN's success. Secure Data, Control, and Application planes will pave the way to the wide adoption of SDN among companies and educational institutions.

With SDN, the network is being opened up as the "brain" of the network is being centralized at the Control Plane. SDN networks are also programmable. Opening up the network and introducing the notion of programmability in SDN is a double-edged sword, as both bring in new opportunities as well as increases attack vectors. As illustrated in Figure 2.2., SDN networks could be attacked from any layer [34].

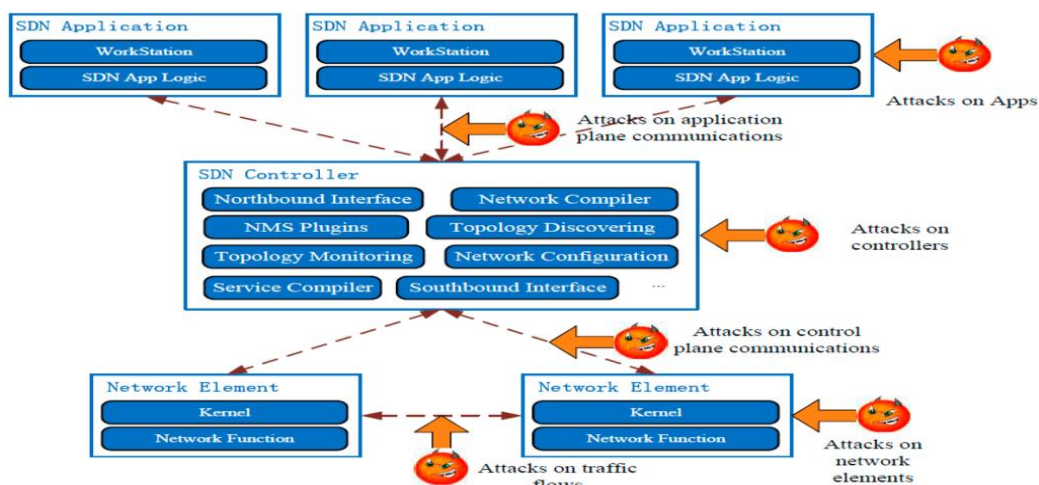


Figure 2.2. Attacks in SDN [34]

At the Application Plane, programmable applications will be introduced. Failure in securely developing those applications could potentially lead to attacks.

The Control Plane is naturally a huge target as it controls the whole network. One of the issues is that the Controller could be a single point of failure, as in, if the controller fails then the whole network fails as well. By compromising the controller, the attacker could change the network traffic behavior [35]. Therefore, a highly fault-tolerant Control Plane is needed [36],[37].

Last but not least, an attacker can compromise the data plane and be able to sniff the packets that go through the network.

All of the above-mentioned facts lead to the need of securing Software Defined Networks.

### 2.3. Network Anomaly Detection: A Machine Learning Perspective

Dr. Dhruva Kumar Bhattacharyya (Tezpur University) and Dr. Jugal Kumar Kalita (University of Colorado) wrote an excellent book titled: “Network Anomaly Detection: A Machine Learning Perspective” [38]. We wrote a review of the book. The authors mentioned that attack patterns might not be easy to find; this is where machine learning comes in. Because machine learning detection techniques are based on behavior and not specific patterns, they give us the ability to detect even unknown attacks. Some Machine Learning techniques used to detect network anomalies are: Supervised Learning, Unsupervised Learning, Probabilistic Learning, Soft Computing, and Combination Learners. From a machine learning perspective, network anomaly detection could be considered as a classification problem since the goal is to classify the traffic as normal or malicious. Existing network anomaly classification methods work in one these four modes: supervised anomaly detection, unsupervised anomaly detection, semi-supervised anomaly detection, and hybrid anomaly detection. As a side note, the authors also discussed some attack-related tools.



Figure 2.3. Taxonomy of Attack Related Tools [38]

### 2.4. Literature Review

Different attempts have been done to detect anomalies in SDN using Machine Learning. One of the most popular example is a paper written by Braga *et al.*[39] in which they proposed a lightweight DDoS flooding attack detection using NOX/OpenFlow. However, NOX is a deprecated controller and is no longer used

nowadays. Figure 2.4 demonstrates the detection loop they used in their implementation. The IDS was developed as a NOX application. It was made up of a flow collector, feature extractor and a classifier. The flow collector periodically collects flows from the OpenFlow switches. Six features are then extracted and given as input to the Self Organizing Maps (SOM) algorithm to do the classification. These features are: Average of Packets per flow (APf), Average of Bytes per flow (ABf), Average of Duration per flow (Adf), Percentage of Pair-flows (PPf), Growth of Single flows (GSf), and Growth of Different Ports (GDP). If an attack is then detected, an alert will be triggered.

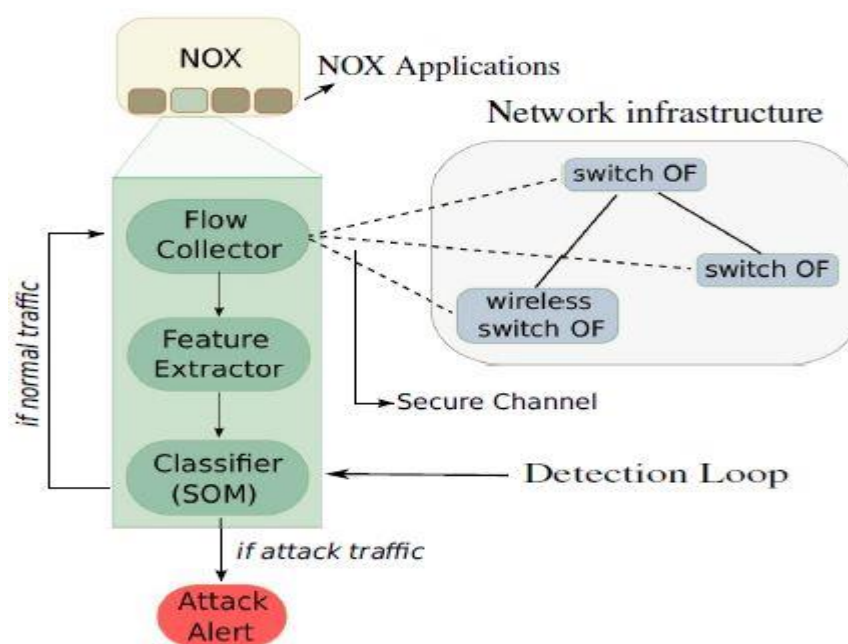


Figure 2.4. Detection Loop Operation [39]

Abubakar, A. and Pranggono, B. proposed a machine learning based intrusion detection system in SDN. However, they haven't implemented the algorithm into the SDN structure and left it as a future work. Dongsoo Lee wrote his thesis on Improving Detection Capability of flow-based IDS in SDN [40]. In fact, part of our work is based on his thesis. Nonetheless, the author didn't provide the machine learning algorithm, thus we ended up building our own algorithm to detect DDoS attacks in the network. Moreover, instead of using the KDD99 dataset which is a very old and outdated dataset, we used flow statistics from the switches in our network and made a dataset out of them. We then trained our algorithm with that data and used it to do the anomaly detection.

## **CHAPTER 3. METHODOLOGY**

We integrated a flow logger module that logs flow statistics received from the OpenFlow switches. Three features (Duration, Packet Count, Byte Count) are then extracted from the statistics and sent to the machine learning algorithm. The algorithm then decides either this is a benign or a malicious flow. If it is malicious, the controller will be triggered and the attacker will be blocked from sending any more traffic into the network.

Machine learning has benefited a lot of fields in science, and it is about time to make use of it in computer networks. By using a classifier, we could train an algorithm to classify traffic flows as either benign or malicious.

### **3.1. Intrusion Detection**

In computer networking, intrusion detection is a technology that allows us to detect suspicious activities and trigger an alarm when such activities are detected. Intrusion detection systems can be classified in terms location or detection methods as shown in figure 3.1. Intrusion detection systems in general have the same basic goal: detect intrusions and alert the network administrator in some way. If they are able to prevent the detected attacks, then they are called Intrusion Detection and Prevention Systems (IDPS).



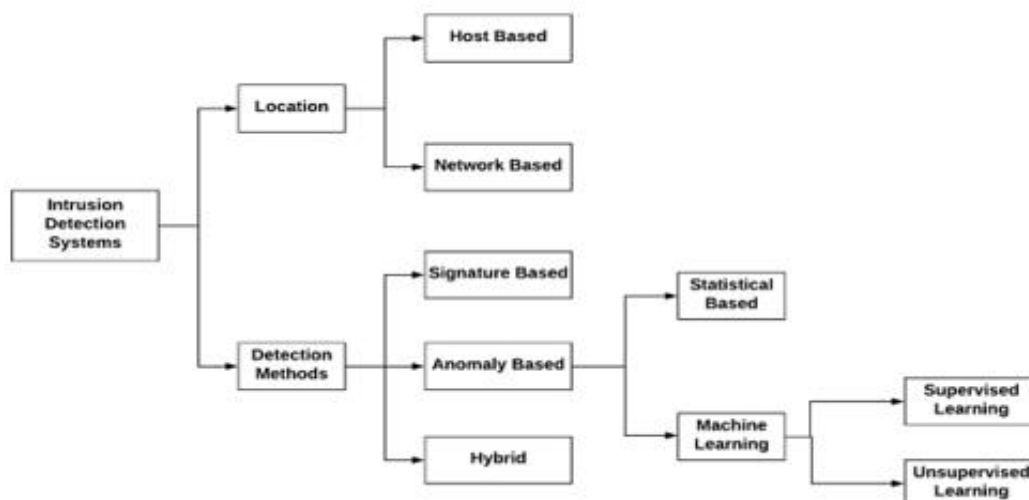


Figure 3.1. Intrusion Detection Systems: A Taxonomy

A host-based IDS is installed directly into a host whereas a network-based IDS is placed in the network to have a global view and detect attacks that target the network as a whole.

Signature-based IDSs are based on specific attack patterns and are very good at detecting known attacks. They also have low false positive rates. However, they have a hard time detecting zero-day attacks. One of the most popular open-source signature-based IDSs is snort.

Anomaly-based IDSs base their detection on behaviors. They build-up a normal behavior for the network and report any unusual or suspicious activities.

A hybrid IDS is one that combines both signature-based and anomaly-based techniques.

### 3.2. Controller of Choice: POX

SDN offers a wide range of controllers. They range from commercial to open source ones. POX is a python based open source SDN controller. Because we developed our algorithm using python scikit-learn, we preferred using a python-based controller. POX comes by default with the Mininet VM and needs no extra setup to make it

work. Using POX, we can create our own modules. It works perfectly with OpenFlow which is the primary and most well-known southbound API [41].

### 3.3. OpenFlow Protocol

One of the first SDN standards that existed was OpenFlow. It is the communication protocol between the control plane and the data plane. Devices need to support the OpenFlow protocol in order to communicate in an OpenFlow environment. Some of the benefits of OpenFlow are programmability and centralized intelligence. Programmability enables innovation by accelerating the integration of new features and services.

#### 3.3.1. OpenFlow Switch

Figure 3.2. depicts the main components of an OpenFlow switch.

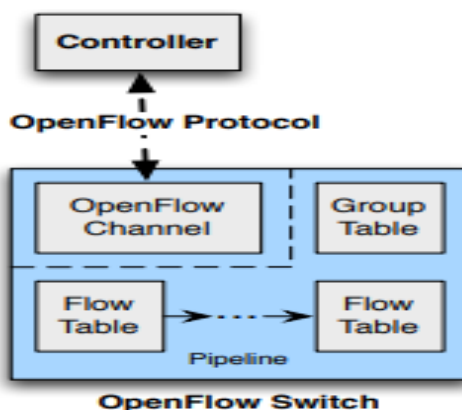


Figure 3.2. Main Components of an OpenFlow Switch [42]

The OpenFlow channel is an interface between the switch and a controller. In an OpenFlow switch, there is one or more flow tables and a group table.

### 3.3.2. Flow Table

In a flow table there are flow entries. The components of a flow entry are:

- match field
- priority
- counters
- instructions
- timeouts
- cookies

### 3.3.3. Flow Stats

We collect flow statistics from the switches periodically and extract features to be given as input to the machine learning algorithm. A request using `ofp_flow_stats_request()` is sent to all switches connected to the controller. The switches reply by sending the flow statistics. The information that is a response to a flow stats request is as follows:

Attribute	Description
ingress port	Length of action entry
ether source	MAC address of source
ether dest	MAC address of the destination
VLAN id	VLAN id of flow
VLAN priority	VLAN priority of flow
IP src	IP address of source
IP dest	IP address of the destination
IP proto	IP protocol
IP ToS bits	Type of service of IPv4
source port	TCP, UDP source port and ICMP Type
dest port	TCP, UDP destination port and ICMP Code

Figure 3.3. Structure of Match Information [40]

Attribute	Description
length	Length of action entry
table_id	ID of match table flow came from
match	Match information of flow
duration_sec	Time flow has been alive in seconds
duration_nsec	Time flow has been alive in nanoseconds
priority	Priority of the entry
idle_timeout	Number of seconds idle before expiration
hard_timeout	Number of seconds before expiration
packet_count	Number of packets in flow
byte_count	Number of bytes in flow

Figure 3.4. Structure of Flow Information [40]

### 3.4. Data Collection

Based on their sources, datasets can be classified into three groups: Public Datasets, Private Datasets, and Network Simulation Datasets [38]. Network simulation datasets are created by simulating normal and attack traffic by considering various attack scenarios. In this project, we are using a network simulation dataset which we created by extracting three features (Duration, Packet Count, and Byte Count) from the flow statistics sent by the switches. The biggest advantage of our approach is that we can easily update our algorithm by simulating new attacks and retraining the algorithm with the new dataset.

### 3.5. Training the Machine Learning Algorithm

We trained the machine learning algorithm on a Ubuntu 14 machine. We compared the accuracy of two algorithms and ended up using the K Nearest Neighbors algorithm since it had a better accuracy than the decision trees algorithm.

```

root@mininet-vm:~/pox/ext/my_ids# sudo python latestMLAlgo.py
KNearestNeighbors : 0.999086866762
DecisionTree : 0.928537398761
The winner is: KNearestNeighbors

*****
Detailed classification report:

confusion matrix
[[11713   23]
 [    0 13452]]
False positive rate : 0.195978 %
False negative rate : 0.000000 %
*****
Accuracy: 0.99881
Precision: 0.99889
Recall: 0.99872
root@mininet-vm:~/pox/ext/my_ids# █

```

Figure 3.5. Training the Machine Learning Algorithm

### **3.5.1. Python Scikit-learn**

Scikit-learn is a simple and efficient python library for data analysis and Machine Learning. It is built on NumPy, SciPy, and Matplotlib. Using Scikit-learn, we can do classification, regressions, clustering, data preprocessing, and many other Machine Learning related tasks. As of October 2017, version 0.19.1 was available for download. Scikit-learn was used for data preprocessing, training, testing, and model persistence. Model persistence is a technique used to save a trained algorithm in a format that would allow us to import it later on for predictions without the need to retrain the algorithm.

### **3.5.2. Decision Trees Algorithm**

Decision Trees (DTs) are supervised learning algorithms used for both classification and regression problems. They are easy to understand and interpret, can be visualized, and can handle both numerical and categorical data.

Some DT algorithms are ID3, C.45, C5.0 and CART. Ross Quilan developed ID3 in 1986. To construct a tree, the algorithm divides attributes into two groups: the most dominant and the others. Entropy and information gains of each attribute is then calculated. The latest version release of Quinlan's is C5.0 which is more accurate. Scikit-learn supports the Classification and Regression Trees (CART). Leo Breiman, a statistician at the University of California, Berkeley contributed greatly in the development of this algorithm. As its names indicates, this algorithm is capable of doing both classification and regression. The CART algorithm provides a basis for other algorithms such random forests.

One disadvantage of decision trees is that sometimes they could generate over-complex trees that do not generalize the data well.

### 3.5.3. K Nearest Neighbors (KNN)

The KNN roots back to the beginning of the 1950s, with Fix and Hodges' work on pattern classification [43]. The K Nearest Neighbors is one of the simplest algorithms and the value of K has a huge influence on the classification problem. Generally, one of three formulas is used for calculating the distance between the data points. These are: The Euclidean Distance, the Manhattan Distance, and the Minkowski Distance. The Euclidean distance is the distance between two points in a Euclidean space. It is calculated as follows [44]:

Euclidean

$$\sqrt{\sum_{i=1}^k (x_i - y_i)^2}$$
(3.1)

Where x and y are the two points and k is the number of neighbors to choose. If K is equal to three for example, then the new data point will be compared to the three closest neighbors and will be classified in the class of the two closest ones.

### 3.5.4. Confusion Matrix

It is a table that allows us to measure the performance of a classifier. It is generally represented as follows:

		Actual Value (as confirmed by experiment)	
		positives	negatives
Predicted Value (predicted by the test)	positives	<b>TP</b> True Positive	<b>FP</b> False Positive
	negatives	<b>FN</b> False Negative	<b>TN</b> True Negative

Table 3.1. Confusion Matrix (General)

The True Positive (TP) is the representation of data points that have been correctly classified as malicious. The True Negative (TN) represents data points or flows that have been correctly classified as benign. The False Positive (FP) indicates the amount of misclassified flows. Last but not least, the False Negative (FN) shows the number of malicious flows that couldn't be detected by the algorithm. Below is the confusion matrix derived from our algorithm.

```

confusion matrix
[[11713    23]
 [    0 13452]]

```

Table 3.2. Confusion Matrix (Our Algorithm)

As we can see, there is no false negative observations in our algorithm.

### 3.5.5. Accuracy

Accuracy is the measurement of correctly predicted data points. It is the sum of the correctly classified flows divided by the total number of flows. Our algorithm showed an accuracy of 0.99%.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.2)$$

It is worth noting that despite this metric being useful, it isn't always enough to confirm the accuracy of a classifier. Other metrics such as the Cumulative Accuracy Profile Curve [45].

### 3.6. DDoS Flood Attack: Detection and Mitigation

A DDoS attack is an attempt by the attacker to consume the resources of the victim until it becomes unavailable. This type of attack is one of the most well known attacks; however, there is still no effective solutions to stop them completely.

A distributed denial of service (DDoS) attack contains three main parts as shown in Figure 3.6. The attacker first selects a group of vulnerable systems (zombies) and installs attack systems in them. The attacker will then launch attack commands to the zombies using a secure channel to carry out the DDoS attack on the target, making it more difficult to trace the origin of the attack.

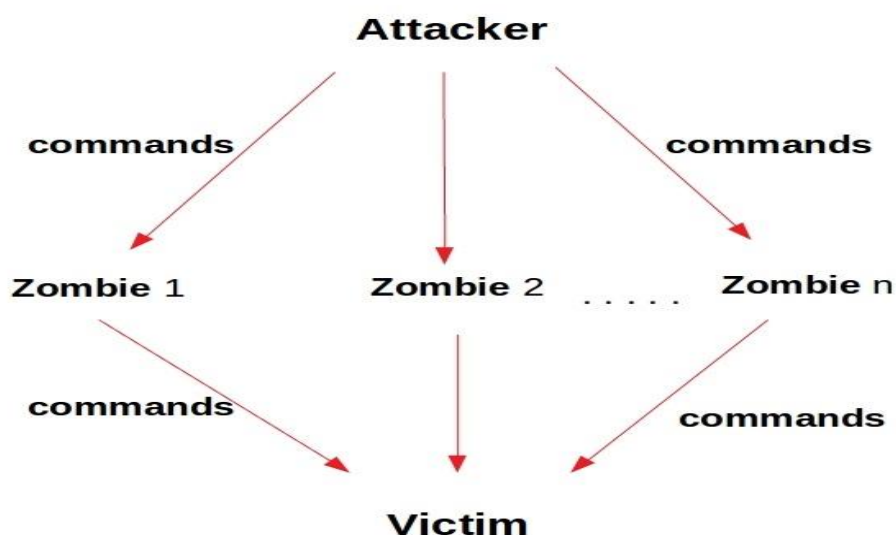


Figure 3.6. DDoS Attack Structure

### 3.7. Limitations of the Project

Our algorithm has been trained to detect ping flood attacks only. Therefore, as of now, it is not able to detect other types of attacks. Besides, the algorithm should be implemented in a real-world environment to analyze its behavior.



## **CHAPTER 4. EXPERIMENTAL IMPLEMENTATION**

In this chapter, we will present the experimental implementation of the project.

### **4.1. Tools used for the Testbed**

Several tools have been used to implement this project.

#### **4.1.1. Mininet**

Mininet [46] is a network emulation tool for rapid prototyping of software defined networks. With a single command, Mininet allows us to create a virtual network running real kernel. This tool is useful for teaching and research. It is probably the best way to learn about SDN and the OpenFlow protocol. With it, we can easily interact with our network using an API or the Command Line Interface (CLI). Briefly it is a fast tool that runs real programs and allows the customization of packet forwarding. It can be run on a laptop in a virtual machine.

Mininet has its limitations as well. These include resource consumption, usage of a single Linux kernel for all virtual hosts, and isolation from our LAN and internet (We may use NAT to connect Mininet to our LAN).

#### **4.1.2. POX Controller**

In SDN, a controller is the brain of the network. Switches have no intelligence in them and receive instructions from controllers. As our machine learning algorithm is developed in python, we decided to use the python based POX controller [47] which

comes by default with Mininet. The POX controller and Mininet work perfectly together. POX was inspired by the NOX controller which is currently deprecated.

POX can't support python3 and works only with python2.7. It also supports OpenFlow1.0 only.

### 4.1.3. Our Machine Learning Algorithm

We built the algorithm based on python Scikit-learn. The algorithm was trained using the dataset that we created earlier.

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import pickle
import sklearn
from sklearn.cross_validation import train_test_split
from sklearn import metrics
from sklearn.metrics import (precision_score, recall_score, f1_score,
                             accuracy_score, mean_squared_error, mean_absolute_error)
```

Figure 4.1. Import Libraries

We start off by importing the necessary libraries. As we can see, most of the libraries are part of sklearn.

```
col_names = ["Duration", "Packet Count", "Byte Count", "label"]
dataset = pd.read_csv('DatasetShuffledCopy.csv', header=None, names = col_names)
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, 3]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

Figure 4.2. Dataset

We import the dataset through pandas (pd). We then split it into training and testing sets. By precising the test set size as 0.25 (25% of the whole dataset), pandas automatically allocates the remaining 75% of the data to the training set.

```

'''Classification'''
from sklearn import tree

model = { "DecisionTree":tree.DecisionTreeClassifier(criterion='entropy', max_depth=5,
                                                    splitter="random"
                                                    ),
          "KNearestNeighbors":KNeighborsClassifier(n_neighbors=3)
        }

results = {}
for algo in model:
    clf = model[algo]
    clf.fit(X_train,y_train)
    score = clf.score(X_test,y_test)
    print ("%s : %s " %(algo, score))
    results[algo] = score

winner = max(results, key=results.get)
print("The winner is: " + winner)

clf = DecisionTreeClassifier(criterion='entropy', max_depth=5, splitter="random")
clf = model[winner]
res = clf.predict(X_test)
expected = y_test
predicted = clf.predict(X_test)
cm = metrics.confusion_matrix(expected, predicted)

```

Figure 4.3. Classification

The classifiers we chose for this project are the K Nearest Neighbors (KNN) and decision trees. We compared the two algorithms and the KNN had the highest accuracy. One of the biggest advantages of Scikit-learn is that classifiers can easily be changed with a single line of code. If we want to, let's say, train our algorithm with the Random Forest algorithm, we can just import its library and add it to the model variable.

```

# make predictions
expected = y_test
predicted = clf.predict(X_test)
# save predicted data
np.savetxt('C:/Users/Soumaine/Desktop/predictedDT.txt', predicted, fmt='%0ld')

```

Figure 4.4. Making Predictions

Making predictions is as simple as doing classification. We can then save our predictions to a file as can be seen.

```
# summarize the fit of the model
accuracy = accuracy_score(expected, predicted)
recall = recall_score(expected, predicted, average=None)
precision = precision_score(expected, predicted, average=None)
f1 = f1_score(expected, predicted, average=None)
cm = metrics.confusion_matrix(expected, predicted)
print(cm)
```

Figure 4.5. Evaluating the Model

The model can then be evaluated to get an idea of how well it would perform later on. These evaluation criteria will be discussed in greater details.

```
from sklearn.externals import joblib
joblib.dump(clf, 'Flearn.pkl')
```

Figure 4.6. Model Persistence

Using joblib, the model can be persisted. The goal of model persistence is to save the model in a format that will allow us to import it to our particular project.

## 4.2. Environment

Mininet is used to create the virtual environment that will allow us to carry out tests and validate our testbed. In this topology, one host will act as an attacker and another host will act as a victim.

### 4.2.1. Topology

To launch the system, we will need to first launch the POX controller and the IDS Module that will allow us to detect the attacks.

```
mininet@mininet-vm:~/pox$ sudo ./pox.py my_ids.startup
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
[my_ids.monitor] Monitoring Ready
[my_ids.monitor] Starting Anomaly Detection
[core] POX 0.2.0 (carp) going up...
[core] Running on CPython (2.7.6/Nov 23 2017 15:50:55)
[core] Platform is Linux-4.2.0-27-generic-i686-with-Ubuntu-14.04-trusty
[core] POX 0.2.0 (carp) is up.
[openflow.of_01] Listening on 0.0.0.0:6633
```

Figure 4.7. Launching the POX Controller and the IDS Module

The IDS Module starts monitoring the network for attacks as soon as the controller is launched. The controller is listening on port 6633.

```
mininet@mininet-vm:~$ sudo mn --controller=remote,port=6633 --topo=tree,depth=2
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet> █
```

Figure 4.8. Launching the Network Topology

In Figure 4.8, we have basically created a tree topology network and connected it remotely to the controller on port 6633. In this topology, we have four hosts, three switches, and one controller.

We now have a functioning network and hosts can ping each other successfully.

```
[openflow.discovery ] link detected: 00-00-00-00-00-01.1 -> 00-00-00-00-00-02.3
[my_ids.monitor ] Sent 3 flow stats request(s)
[my_ids.monitor ] FlowStatsReceived from 00-00-00-00-00-01 : 42
[my_ids.monitor ] FlowStatsReceived from 00-00-00-00-00-03 : 52
[my_ids.monitor ] FlowStatsReceived from 00-00-00-00-00-02 : 52
[openflow.discovery ] link detected: 00-00-00-00-00-01.2 -> 00-00-00-00-00-03.3
[my_ids.monitor ] Sent 3 flow stats request(s)
[my_ids.monitor ] FlowStatsReceived from 00-00-00-00-00-01 : 42
[my_ids.monitor ] FlowStatsReceived from 00-00-00-00-00-03 : 52
[my_ids.monitor ] FlowStatsReceived from 00-00-00-00-00-02 : 52
```

Figure 4.9. Flow Stats

The IDS sends flow stats requests using POX's `ofp_flow_stats_request` messages and the three switches reply by sending their flow statistics. Three features (Duration, Packet Count, Byte Count) are then extracted and forwarded to the flow IDS module. This module then checks the flow and decides if it is benign or malicious.

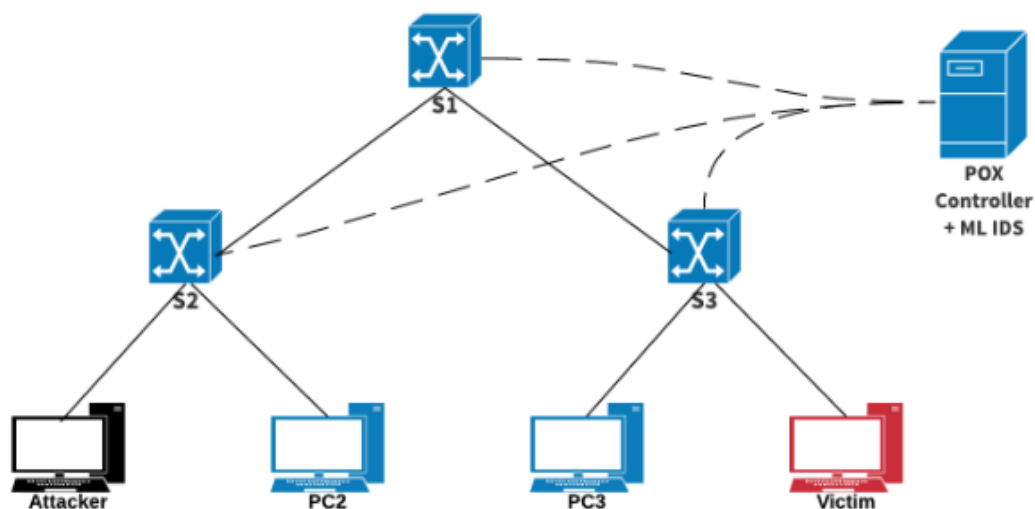


Figure 4.10. Topology Created in Mininet

#### 4.2.2. Attack Scenario

To carry out a ping flooding attack, the attacker will send tens of thousands of packets per second to the victim, flooding it with unnecessary echo request packets. Figure 4.11. illustrates a sequence diagram, where it explains the attack scenario and how the testbed deals with ping flooding attacks. In the sequence diagram shown below, the attacker sends a normal ping and it goes to the OpenFlow enabled switch. The switch checks the flow table, and because the destination is unknown to the switch (table miss), it forwards the packet to the SDN controller (packet-in) to ask for the destination of the packet. The SDN controller replies with a packet-out which includes information needed by the switch, then the switch stores the destination of the route into its flow table. The attacker then launches the DoS attack. When the controller detects the attack, it will send an alert and block the attacker from sending any more packets into the network.

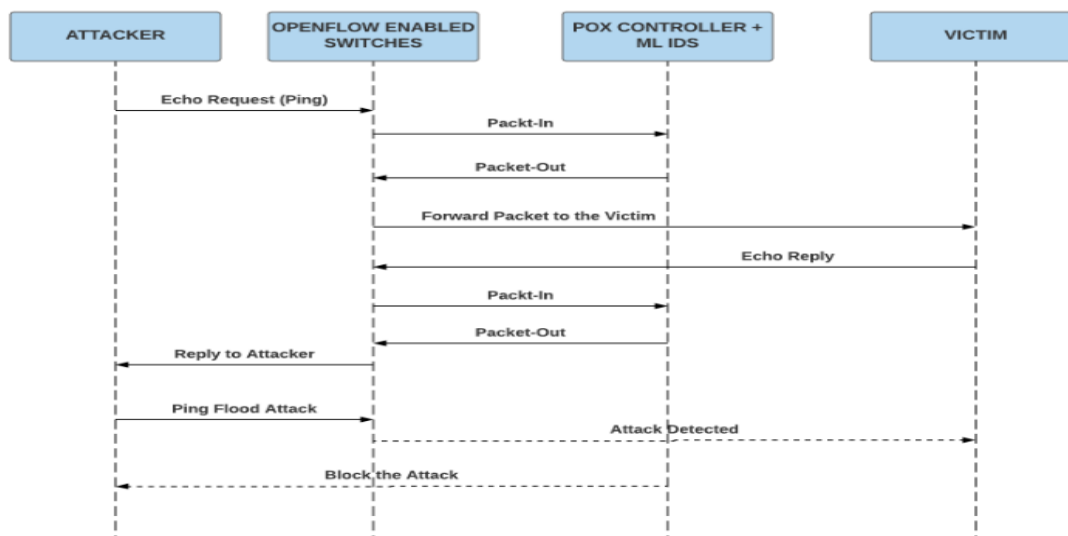


Figure 4.11. Sequence diagram

### 4.3. Results

After setting up and running the environment, we execute a ping flood attack to test our algorithm. As seen in the screen shots, as soon as the flooding begins, the IDS shows in its log messages that a malicious flow has been detected and the POX controller is triggered to take an action and drop subsequent packets from the attacking host.

```

my_ids.forwarding ] installing flow for 3e:97:fe:94:6a:f4.3 -> 76:ec:c9:cd:90:82.2
my_ids.forwarding ] installing flow for 76:ec:c9:cd:90:82.2 -> 3e:97:fe:9
my_ids.forwarding ] installing flow for 76:ec:c9:cd:90:82.3 -> 3e:97:fe:9
my_ids.monitor    ] Sent 3 flow stats request(s)
my_ids.monitor    ] FlowStatsReceived from 00-00-00-00-00-01 : 7
my_ids.monitor    ] Extracting Features
my_ids.monitor    ] Malicious flow detected
my_ids.monitor    ] Extracting Features
my_ids.monitor    ] Malicious flow detected
my_ids.monitor    ] FlowStatsReceived from 00-00-00-00-00-03 : 7
my_ids.monitor    ] FlowStatsReceived from 00-00-00-00-00-02 : 7
my_ids.monitor    ] Sent 3 flow stats request(s)
my_ids.monitor    ] FlowStatsReceived from 00-00-00-00-00-01 : 9
my_ids.monitor    ] FlowStatsReceived from 00-00-00-00-00-03 : 9
my_ids.monitor    ] FlowStatsReceived from 00-00-00-00-00-02 : 9
my_ids.monitor    ] Sent 3 flow stats request(s)
my_ids.monitor    ] FlowStatsReceived from 00-00-00-00-00-01 : 9
my_ids.monitor    ] FlowStatsReceived from 00-00-00-00-00-03 : 9
my_ids.monitor    ] FlowStatsReceived from 00-00-00-00-00-02 : 9
my_ids.monitor    ] Sent 3 flow stats request(s)
my_ids.monitor    ] FlowStatsReceived from 00-00-00-00-00-01 : 7
my_ids.monitor    ] FlowStatsReceived from 00-00-00-00-00-03 : 7
my_ids.monitor    ] FlowStatsReceived from 00-00-00-00-00-02 : 7
  
```

```

root@mininet-vm:~# ping 10.0.0.4 -f
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data:
.....
^C
--- 10.0.0.4 ping statistics ---
1283 packets transmitted, 0 received, 100% packet loss, time 15430ms

root@mininet-vm:~# ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data:
^C
--- 10.0.0.4 ping statistics ---
12 packets transmitted, 0 received, 100% packet loss, time 11080ms

root@mininet-vm:~#
  
```

Figure 4.12. Ping Flood Attack Detection and Mitigation

## **CHAPTER 5. CONCLUSION AND FUTURE WORK**

In this project, we proposed an IDS that allows us to detect ping flood attacks in an SDN network. The results show that choosing a network simulation dataset has its advantages as it allowed us to detect attacks. Machine learning with Scikit-learn gives us more flexibility as the algorithm can be changed in literally two lines of code. The dataset can also be easily updated to try other or new types of attacks.

Mininet is a powerful tool and probably the best way to learn about SDN. Through it, we can emulate networks and connect them to any types of controllers in no time.

Machine learning has its benefits, and for SDN to become more powerful and flexible, more and more machine learning algorithms should be tested and implemented in SDN.

Overall, despite POX still supporting only python2.7, it can also help in developing IDSs. However, the developers of POX should consider updating it to python3 and include support for newer versions of OpenFlow (as of this writing, POX still supports OpenFlow 1.0 only).

As part of future work, new attack vectors could be implemented and tested as well. Implementing the algorithm in a hybrid environment, such as implementing directly into snort would be very interesting. This way, we will be able to do signature-based and anomaly-based intrusion detection simultaneously.



## REFERENCES

- [1] Jonathan Crane, "Outage Prevention: Taking Humans Out Of The IT Equation," *forbes*, 2012. [Online]. Available: <https://www.forbes.com/sites/ciocentral/2012/10/22/outage-prevention-taking-humans-out-of-the-it-equation/#3603b7504dd1>, Access Date: 09-Oct-2017.
- [2] Kathleen Hickey, "What's behind most data center outages? -- GCN," 2016. [Online]. Available: <https://gcn.com/articles/2016/02/09/data-center-outages.aspx>, Access Date: 09-Oct-2017.
- [3] Press Release, "Global Survey: Complexity, Change and Human Factors Cause Network Outages - The Data Center Journal," 2016. [Online]. Available: <http://www.datacenterjournal.com/global-survey-complexity-change-human-factors-cause-network-outages/>, Access Date: 09-Oct-2017.
- [4] J. Networks Inc, "What's Behind Network Downtime? Proactive Steps to Reduce Human Error and Improve Availability of Networks," 2008.
- [5] Rachel King, "Amazon Web Services Outage Caused by Human Error: A Typo | Fortune," 2017. [Online]. Available: <http://fortune.com/2017/03/02/amazon-cloud-outage/>, Access Date: 09-Oct-2017.
- [6] E. Description, "Cisco Certified Network Associate," 2016.
- [7] "Software-Defined Networking (SDN) Definition - Open Networking Foundation." [Online]. Available: <https://www.opennetworking.org/sdn-definition/>, Access Date: 13-Jun-2018.
- [8] "Single Point of Failure. Not. - Open Networking Foundation." [Online]. Available: <https://www.opennetworking.org/news-and-events/blog/single-point-of-failure-not/>, Access Date: 05-Jun-2018.
- [9] "SDN FAQ | Network World." [Online]. Available: <https://www.networkworld.com/article/2167706/lan-wan/lan-wan-sdn-faq.html>, Access Date: 05-Jun-2018.
- [10] A. Mestres *et al.*, "Knowledge-Defined Networking," Jun. 2016.

- [11] “OpenDaylight Summit 2016: OpenDaylight Machine Learning & Artificial...” [Online]. Available: <https://opendaylightsummit2016.sched.com/event/80Nz>, Access Date: 05-Jun-2018.
- [12] “Machine-Learning for Network Security: There’s an App for That.” [Online]. Available: <https://gab41.lab41.org/machine-learning-for-network-security-theres-an-app-for-that-e9bc01139f19>, Access Date: 05-Jun-2018.
- [13] N. Feamster, J. Rexford, and E. Zegura, “The Road to SDN: An Intellectual History of Programmable Networks,” *ACM Sigcomm Comput. Commun.*, vol. 44, no. 2, pp. 87–98, 2014.
- [14] C. D. Richard, *Broadcasting and Optical Communication Technology - Richard C. Dorf - Google Books*, Third Edition. California: Taylor & Francis Group, 2006.
- [15] N. Feamster, “Software Defined Networking,” *Georg. Inst. Technol.*, p. 11.
- [16] Margaret Rouse, “What is active network? - Definition from WhatIs.com,” 2005. [Online]. Available: <http://searchnetworking.techtarget.com/definition/active-network>, Access Date: 10-Oct-2017.
- [17] N. Feamster, J. Rexford, and E. Zegura, “The road to SDN,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, 2014.
- [18] J. M. Smith, “Reflections on Active Networking,” Pennsylvania, 2003.
- [19] K. L. Calvert, S. Bhattacharjee, E. Zegura, and J. Sterbenz, “Directions in active networks,” *IEEE Commun. Mag.*, vol. 36, no. 10, pp. 72–78, 1998.
- [20] W. David, Andrew Whitaker, and Jan, “Active Networks at UW,” *University of Washington*. [Online]. Available: <http://research.cs.washington.edu/networking/ants/>, Access Date: 10-Oct-2017.
- [21] University of Pennsylvania, “The SwitchWare Project.” [Online]. Available: <http://dsl.cis.upenn.edu/switchware/>, Access Date: 10-Oct-2017.
- [22] B. E. Schwartz Al *et al.*, “Smart Packets for Active Networks.”
- [23] A. T. Campbell, I. Katzela, K. Miki, and J. Vicente, “Open signaling for ATM, internet and mobile networks (OPENSIG’98),” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 1, p. 97, 1999.

- [24] S. Rooney, J. E. van der Merwe, S. A. Crosby, and I. M. Leslie, "The Tempest: a framework for safe, resource assured, programmable networks," *IEEE Commun. Mag.*, vol. 36, no. 10, pp. 42–53, 1998.
- [25] SDxCentral, "What is Network Virtualization? - Definition," 2017. [Online]. Available: <https://www.sdxcentral.com/sdn/network-virtualization/definitions/whats-network-virtualization/>, Access Date: 10-Oct-2017.
- [26] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI Veritas: Realistic and Controlled Network Experimentation," *ACM Sigcomm Comput. Commun.*, p. 12, 2006.
- [27] H. Khosravi, "RFC 3654 - Requirements for Separation of IP Control and Forwarding," 2003.
- [28] J. Hadi Salim, Z. R. Haas, E. H. IBM Khosravi, E. W. Intel Wang, and E. L. Dong, "Forwarding and Control Element Separation (ForCES) Protocol Specification," 2010.
- [29] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe, "The case for separating routing from routers," *Proc. ACM SIGCOMM Work. Futur. Dir. Netw. Archit. - FDNA '04*, p. 5, 2004.
- [30] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane," in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '07*, 2007, vol. 37, no. 4, pp. 1–12.
- [31] B. et al. Heller, "OpenFlow Switch Specification 1.5.1," 2015.
- [32] SDxCentral, "Special Report: OpenFlow and SDN – State of the Union," 2016.
- [33] "Open Networking Foundation is an operator led consortium leveraging SDN, NFV and Cloud technologies to transform operator networks and business models." [Online]. Available: <https://www.opennetworking.org/>, Access Date: 11-Oct-2017.
- [34] S. Luo, M. Dong, K. Ota, J. Wu, and J. Li, "A Security Assessment Mechanism for Software-Defined Networking-Based Mobile Networks," *Sensors*, vol. 15, no. 12, pp. 31843–31858, Dec. 2015.
- [35] "SDN security issues: How secure is the SDN stack?" [Online]. Available: <http://searchsdn.techtarget.com/news/2240214438/SDN-security-issues-How-secure-is-the-SDN-stack>, Access Date: 04-Nov-2017.

- [36] A. J. Gonzalez, G. Nencioni, B. E. Helvik, and T. Asa, "A Fault-Tolerant and Consistent SDN Controller," 2016.
- [37] M. Azab and J. A. B. Fortes, "Towards proactive SDN-controller attack and failure resilience," *2017 Int. Conf. Comput. Netw. Commun. ICNC 2017*, pp. 442–448, 2017.
- [38] J. Kalita, *Network Anomaly Detection*. 2013.
- [39] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS Flooding Attack Detection Using NOX / OpenFlow Network-Based Mechanisms Using SDN Network-Based Mechanisms Using SDN," pp. 408–415, 2018.
- [40] L. Dongsoo, "Improving Detection Capability of Flow-based IDS in SDN," 2015.
- [41] "What are SDN Southbound APIs? - Where they are used." [Online]. Available: <https://www.sdxcentral.com/sdn/definitions/southbound-interface-api/>, Access Date: 02-Jun-2018.
- [42] "OpenFlow Switch Specification," 2015.
- [43] L. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.
- [44] S. Sayad, "K Nearest Neighbors."
- [45] Alaa Khaled, "Classification Models Performance Evaluation — CAP Curve," 2017. [Online]. Available: <https://medium.com/@lotass/classification-models-performance-evaluation-c3a91562793>, Access Date: 03-Jul-2018.
- [46] "Mininet." [Online]. Available: <http://mininet.org/download/>. [Access Date: 01-Jun-2018].
- [47] "Installing POX — POX Manual Current documentation." [Online]. Available: <https://noxrepo.github.io/pox-doc/html/#>, Access Date: 01-Jun-2018.

## **RESUME**

Soumaine Boubou Mahamat was born on 01.01.1989 in N'djamena, Chad. He completed his primary school at Lycée de la Réussite in N'djamena, and his secondary school at Lycée Ibnou-Cina in 2008. He then went to the City of Zaria in Nigeria to study English Language in January 2009. The same year he went to Malaysia and started his Bachelor's Degree in Information and Communication Technology at KUIS. He graduated from KUIS in 2012. In 2013, Soumaine got a job offer from Computer Golfe Tchad where he worked as a Network Engineer for almost three years. In July 2015, he received a scholarship offer from YTB (Yurtdışı Türkler ve Akraba Topluluklar Başkanlığı) and came to Turkey to further his Masters studies at Sakarya University. Soumaine holds multiple certificates such as the Cisco CCNA which he received on July 17, 2017 and a Fiber Optics Installation Technician Certificate. He has been socially very active since his days in Malaysia. He used to be the Representative of International Students at the IT Department in KUIS. He was also the Chairman of the Welfare Committee of AFROKARYA (Sakarya African Students Association) and the representative of Chadian students in Sakarya at UDETTTC (Union of Chadian Students in Turkey and Cyprus). He was responsible of Communication at UDETTTC and is currently its Head of Academic and Social Affairs.