

People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Saad Dahleb University of Blida 1
Faculty of Sciences
Computer Science department



MASTER THESIS
In computer science

Option : - Computer Systems and Networks
- Information Systems Security

**Detection and mitigation of DDoS attacks
in SDN networks.**

Realized by:

- Amani Alili
- Yasmine Boudarene

The jury:

- Dr. Sana Aroussi (President)
- Dr. Afrah Djeddar (Examinator)
- Dr. Zakaria Sahnoune (Promotor)

Promotion : 2020/2021

Acknowledgement

First and foremost, we thank GOD, the Almighty, for assisting us along the path and providing us with the strength and ability to complete this memory.

Secondly, we would like to thank our promoter, Dr. Zakaria Sahnoune, for the confidence he has placed in offering us this research theme, and his accompaniment, support, and availability for the realization of our work.

We sincerely thank our dear parents, families and all our friends for their love and encouragement.

ملخص

الشبكات المعرفة بالبرمجيات (SDN) هي مفهوم ناشئ مصمم ليحل محل الشبكات التقليدية عن طريق كسر التكامل الرأسي. التحكم المركزي هو أكبر نقطة قوة لـ SDN ، لكنه يمثل نقطة فشل خاصة بالنسبة لهجمات حجب الخدمة الموزعة (DDoS) التي تجعله غير قابل للوصول و غير متاح. توفر هذه الأطروحة حلاً فعالاً يعتمد على خوارزميات التعلم الآلي (ML) لاكتشاف هجمات DDoS والتخفيف من حدتها باستخدام وحدة تحكم Mininet و Ryu لمحاكاة الشبكة، بينما تمت محاكاة هجمات DDoS باستخدام أداة Hping3. تثبت هذه الأطروحة أن نوع الهيكل مهم ويمكن أن يؤثر على النسبة المئوية لنجاح هجمات DDoS في مثل هذه الشبكات. تم اختبار وتقييم ست خوارزميات خاضعة للإشراف (LR و K-NN و NB و SVM و DT و RF) باستخدام مجموعة بيانات تركيبية. أظهرت النتائج أن DT و RF هما الأفضل مقارنة بالخوارزميات الأخرى بدقة 100٪. يُظهر النظام المقترح كفاءته في اكتشاف هجمات DDoS والتخفيف من حدتها باستخدام المصنف RF وخمس ميزات فقط بينما تم توفير التخفيف عن طريق إضافة قاعدة تدفق إلى المحول لقمع حركة المرور الضارة.

الكلمات المفتاحية :

الشبكات المعرفة بالبرمجيات (SDN)، هجوم رفض الخدمة (DDoS)، التعلم الآلي (ML).

Abstract

Software-Defined Networking (SDN) is an emerging concept designed to substitute traditional networking by breaking up vertical integration. Central control is the biggest benefit of SDN, but a single point of failure is also a failure if a distributed denial of service (DDoS) attack makes it unattainable. This memory provides an efficient solution based on machine learning (ML) algorithms to detect and mitigate DDoS attacks with the help of Mininet and the Ryu controller to simulate the network. In contrast, DDoS attacks were simulated using of Hping3 tool. This memory proves that the type of topology is significant and can affect the percentage of success of DDoS attacks in such networks. Six supervised ML algorithms (LR, K-NN, NB, SVM, DT, and RF) were tested and evaluated using a synthetic dataset. The results show that DT and RF are the best compared to the other algorithms with 100% of accuracy. The proposed system shows its efficiency in detecting and mitigating DDoS attacks with the RF classifier and only five features. At the same time, the mitigation was provided by adding a flow rule to the switch to drop the malicious traffic.

Keywords:

Software-Defined Networking (SDN), Distributed Denial of Service Attack (DDoS), Machine learning (ML).

Résumé

Le réseau défini par logiciel ou SDN (Software-Defined Networking) est un concept émergent conçu pour remplacer les réseaux traditionnelles en brisant l'intégration verticale. Le contrôle central est le plus grand avantage du SDN, mais un point de défaillance unique est également un échec si une attaque par déni de service distribué (DDoS) le rend inaccessible. Ce mémoire fournit une solution efficace basée sur des algorithmes d'apprentissage automatique (ML) pour détecter et mitiger les attaques DDoS à l'aide de Mininet et du contrôleur Ryu pour simuler le réseau, tandis que les attaques DDoS ont été simulées à l'aide de l'outil Hping3. Ce mémoire prouve que le type de la topologie est significatif et peut affecter le pourcentage de réussite des attaques DDoS dans tels réseaux. Six algorithmes supervisés (LR, K-NN, NB, SVM, DT et RF) ont été testés et évalués à l'aide d'un ensemble de données synthétiques. Les résultats montrent que DT et RF sont les meilleurs par rapport aux autres algorithmes avec 100% de précision. Le système proposé montre son efficacité pour détecter et atténuer les attaques DDoS avec le classificateur RF et cinq critères uniquement tandis que l'atténuation a été fournie en ajoutant une règle de flux au commutateur pour supprimer le trafic malveillant.

Mots clé:

Mise en réseau définie par logiciel (SDN), Attaque de déni de service (DDoS), Apprentissage automatique (ML).

Contents

General Introduction	14
I Notions of Network Security, SDN, DDoS	17
I.1 Introduction	18
I.2 Network security	18
I.2.1 Definition	18
I.2.2 Objectives	18
I.2.3 Network attacks	19
I.2.4 Network protection	20
I.3 SDN (Software-Defined Network)	21
I.3.1 Definition	21
I.3.2 Comparison between SDN networks and traditional networks	21
I.3.3 The architecture of the SDN	22
I.3.4 SDN controllers	24
I.3.5 Benefits of SDN	25
I.3.6 Challenges of SDN	26
I.4 OpenFlow fundamentals	28
I.4.1 Switch Components	29
I.4.2 Openflow ports	29
I.4.3 Openflow channel	31
I.5 Denial of Service	31
I.5.1 Definition	31
I.5.2 Distributed denial of service (DDoS) Attacks	32
I.5.3 Botnets	32
I.5.4 Taxonomy of DDoS attacks	32
I.6 Conclusion	35
II DDoS attacks in SDN networks, and ML approach	37
II.1 Introduction	38
II.2 Machine learning approach	39
II.2.1 Introduction to machine learning	39
II.2.2 Types of machine learning algorithms	40

II.3 Supervised machine learning	41
II.3.1 k-Nearest Neighbor (k-NN)	41
II.3.2 Decision Tree	42
II.3.3 Support Vector Machine	43
II.3.4 Random Forest	43
II.3.5 Logistic Regression	44
II.3.6 Naive Bayes	44
II.4 Dataset	45
II.4.1 Types of dataset	45
II.5 Data Preprocessing	45
II.5.1 Feature Selection	45
II.5.2 Feature Selection Procedure	46
II.5.3 Feature selection methods	46
II.6 Classifier Evaluation Measures	47
II.6.1 Accuracy	48
II.6.2 Confusion matrix	48
II.7 SDN Security problems	49
II.7.1 Communication Level	49
II.7.2 Each Component level	49
II.7.3 Logging and audit level	50
II.8 Defeating DDoS attacks in SDN based Networks	50
II.9 Types of DDoS attacks in SDN	51
II.9.1 Application layer DDoS attacks	52
II.9.2 Control layer DDoS attacks	52
II.9.3 Data layer DDoS attacks	53
II.9.4 Communication links	54
II.10 Detecting DDoS attacks in SDN network	54
II.10.1 DDoS attack detection techniques	54
II.11 Discussion	57
II.12 Conclusion	57
III Proposed approach and implementation	58
III.1 Introduction	59
III.2 Proposed approach	59
III.3 How the proposed approach works	61
III.4 Material resources	62
III.5 Tools selection	62
III.5.1 Python language	62
III.5.2 Ryu Controller	62

III.5.3 Mininet virtual network	63
III.5.4 Hping the DDoS attack tool	64
III.6 Files and functions used in the implementation	64
III.7 Setup the network	67
III.7.1 Tree topology	67
III.7.2 Linear topology	69
III.8 Dataset and ML algorithms	71
III.8.1 Dataset selection	71
III.8.2 Comparison between ML algorithms	73
III.8.3 Features selection	75
III.8.4 Integration of the model with Ryu controller	78
III.9 Work evaluation	79
III.10 Conclusion	80
Conclusion and Future Work	81
Bibliography	81
Appendix	89
A Deposit of the source code	90

List of Figures

I.1	The architecture of SDN Vs Traditional Network, adopted from [1]	22
I.2	Architecture of SDN, adopted from [2]	23
I.3	Probable Attacks on SDN Architecture, adopted from [3]	28
I.4	Out-of-band and in-band control plane, adopted from [4]	28
I.5	Openflow switch components, adopted from [5]	30
I.6	DDoS attack scenario	33
I.7	DDoS attacks Taxonomy.	34
I.8	TCP SYN flood Scenario, adopted from [6]	35
II.1	SDN architecture and components, adopted from [7]	39
II.2	Feature selection process, adopted from [8]	46
II.3	The filter method, adopted from [8]	47
II.4	The embedded method, adopted from [8]	47
II.5	The wrapper method, adopted from [8]	48
II.6	Types of DDoS attacks at different layers of SDN, adopted from [9]	52
II.7	Classification of DDoS attack defence approaches in SDN, adopted from [9].	55
III.1	The proposed approach.	60
III.2	Proposed approach.	61
III.3	Tree topology.	68
III.4	Linear topology.	70
III.5	Executing monitor.py script as Ryu application.	71
III.6	Generating normal traffic.	72
III.7	Comparison between algorithms.	74
III.8	Comparison between algorithms.	75
III.9	Top five feature importance.	76
III.10	Evaluation with full features.	77
III.11	Evaluation with reduced features.	77
III.12	Running the detector_mitigator.py application.	78
III.13	Results indicating that the capability of the system to detect and mitigate DDoS attacks.	79

List of Tables

I.1	The characteristics of SDN & Traditional networks architecture.	22
II.1	Confusion matrix	48
II.2	DDoS defense solutions in SDN.	56
III.1	Features of Ryu controller	63
III.2	Description of used files.	65
III.3	Description of the used DDoS functions.	66
III.4	Results of pinging from H3 (switch S2) TO H4, H6, H14, H15 and H1.	68
III.5	Results of pinging from h8 (switch S3) to H15, H3, H12, H21, H1, H7, H9.	68
III.6	Results of pinging from H10 (switch S4) to H4, H8, H12, H15, H17, H16.	69
III.7	Results of pinging from H16 (switch S5) to H2, H17, H15, H20, H19	69
III.8	Results of pinging from H1 (switch S3) (switch S1) to H4 and H5 and H6.	70
III.9	Results of pinging from H6 (switch S3) to H4, H5, H3 and H1.	70
III.10	Features description.	73
III.11	Work summary.	79

Source Code

A.1 monitor.py	90
A.2 ML.py	93
A.3 detector_mitigator.py	96

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ARP	Address Resolution Protocol
BGP	Border Gateway Protocol
CHARGEN	Character Generator Protocol
CLI	Command Line Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSV	Character Separated Values
DoS	Denial of Service
DDoS	Distributed Denial of Service
DDS	Data Distribution Service
DL	Deep Learning
DNS	Domain Name System
DPID	Delivery Point Identifier
DT	Decision Tree
EIGRP	Enhanced Interior Gateway Routing Protocol
FML	Flash Markup Language
FN	False Negative
FP	False Positive
FS	Feature Selection
ForCES	Forwarding and Control Element Separation Protocol

HTTP	Hypertext Transfert Protocol
ICMP	Internet Control Message Protocol
IT	Information Technology
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IDPS	Intrusion Detection and Prevention System
IoT	Internet of Things
IP	Internet Protocol
IPS	Intrusion Prevention Systems
ISP	Internet Service Provider
K-NN	k-Nearest Neighbours
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LR	Logistic Regression
LSTM	Long Short-Term Memory
MAC	Media Access Control
ML	Machine Learning
MSSQL	Microsoft Structured Query Language
NAT	Network Address Translation
NB	Naive Bayes
NetBIOS	Network Basic Input Output System
NFG	Network Flow Guard
NOS	Network Operating System
NTP	Network Time Protocol
ONOS	Open Network Operating System
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
POF	Protocol-Oblivious Forwarding
QoS	Quality of Service

RF	Random Forest
RNN	Random Neural Network
SDN	Software-Defined Network
SNMP	Simple Network Management Protocol
SOM	Self-Organized Mapping
SSDP	Simple Service Discovery Protocol
STP	Spanning Tree Protocol
SVM	Support Vector Machine
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
TCL	Tae Command Language
TFTP	Trivial File Transfer Protocol
TLS	Transport Layer Security
TN	True Negative
TP	True Positive
TTL	Time To Live
TTA	Time To acknowledge
UDP	User Datagram Protocol
WAN	Wide Area Network

General Introduction

In the existing networks, traffic flows are sent through network devices such as routers and switches distributed around the world. Network devices are responsible for controlling and propagating traffic. While these traditional networks are widespread, they have some drawbacks. First, it does not provide the flexibility for researchers to experiment, add new features, and protocol [10]. Second, existing networks cannot be programmed and therefore cannot accept new commands to enhance functionality. Third, because each device contains both a control plane and a data plane, the cost of network devices is very high [11]. However, SDN (Software Defined Networking) solves the problems of existing networks. SDN is a programmable, virtualized network that helps you insert new ideas into your research. SDN removes the control plane from the data plane. The control plane is responsible for processing information, while the data plane is responsible for data transfer [3]. SDN can be deployed in many different networks, such as private networks, enterprise networks, and wide area networks. Unfortunately, SDN has many challenges that need to be addressed. Scalability, performance, and security are some of the challenges that face SDN.

The centralized structure of the controller could lead to many security challenges. One of such critical challenges is the impact of distributed denial of service attacks (DDoS) on SDN networks. Such attacks can quickly bring down the entire network by bringing down the controller. Since the attack packets are sent with many spoofed source IPs, the DDoS attack can cause problems to both switches, and the controller [12]. Furthermore, in the flood of DDoS attacks, attackers use many different spoofing source IP addresses, making it impossible to stop attacks by blocking traffic based on the source IP address alone. The practical implementation of DDoS detection and response methods has been the key to the regular operation of the network. This problem is more present in SDN networks due to its single point of failure (controller).

Motivation:

Many companies such as CISCO¹, INTEL², IBM³, HP⁴, HUAWEI⁵, JUNIPER⁶, ORACLE⁷, VMware⁸, Orange⁹, and so on, Establishes the Open Networking Foundation (ONF)¹⁰, develop the OpenFlow specification, and promote the use of SDN. Other technology companies are also part of this association, such as Cisco, Juniper, Broadcom¹¹, Dell¹², IBM, Riverbed Technology¹³, HP, Citrix¹⁴, Netgear¹⁵, Force10¹⁶, and NTT¹⁷ [13]. The market size of SDN is growing day after day; in 2020, the SDN market reached 8 billion U.S. dollars in size, and it is predicted to reach 43 billion U.S. dollars by 2027 [14]. Although the brain of SDN, which is the controller, is still vulnerable to DDoS attacks. In 2020, 69 percent of large enterprises experienced an increase in demand for distributed denial of service (DDoS) attacks. DDoS attacks are a serious threat to businesses by threatening service performance or shutting down a website completely [15].

In this memory, we aim to study the DDoS attacks in SDN networks, available techniques to detect and mitigate this attack, study the impact of DDoS on different topologies, and of course, implement a system that detects and mitigates DDoS in SDN networks.

This work is organized in this way:

Chapter I: gives an overview of the essential concept about network security, SDN networks, OpenFlow, and DDoS attacks in general.

Chapter II: provides more details about DDoS attacks in SDN networks and the available works and research that carried this subject.

¹ <https://www.cisco.com>.

² <https://www.intel.com>.

³ <https://www.ibm.com>.

⁴ <https://www.hp.com>.

⁵ <https://www.huwaei.com>.

⁶ <https://www.jupiner.net>.

⁷ <https://www.oracle.com>.

⁸ <https://www.vmware.com>.

⁹ <https://www.orange.com>.

¹⁰ The Open Networking Foundation is a non-profit operator-led consortium. It uses an open source business model aimed at promoting networking through software-defined networking and standardizing the OpenFlow protocol and related technologies. <https://www.opennetworking.org>

¹¹ Broadcom is a global technology leader that designs, develops and supplies semiconductor and infrastructure software solutions. <https://www.broadcom.com>.

¹² <https://www.dell.com>

¹³ Riverbed Technology is an American information technology company. Its products consist of software and hardware focused on network performance monitoring, application performance management, and wide area networks, including SD-WAN and WAN optimization. <https://www.riverbed.com>

¹⁴ <https://www.citrix.com>

¹⁵ <https://www.netgear.com>

¹⁶ <https://www.force10.com>

¹⁷ <https://www.global.ntt>

Chapter III: an overview of the proposed approach to detect and mitigate DDoS attacks in SDN networks and explain the process of our system to detect and mitigate these attacks.

Chapter IV: represent the implementation of the proposed system, results, and evaluation. In the end, we provide a general conclusion with future works.

Chapter I

Notions of Network Security, SDN, DDoS

I.1 Introduction

This chapter represents an overview of some concepts that we need to go along with this thesis. It defines network security and attacks. It shows the purpose of network security. It explains the difference between traditional networks and SDN networks. It also explains the OpenFlow Fundamentals. Finally, it presents DDoS attacks and countermeasures.

I.2 Network security

I.2.1 Definition

The first question to address is what we mean by "network security." Several possible fields of endeavor come to mind within this broad topic, and each is worthy of a lengthy article. First of all, network security is a subset of computer security [16].

Security in networks starts with physical protection. In terms of the OSI model, there are different levels at which encryption can be done. It can be done at the lowest layers, Physical and Data Link or the higher layers, such as Network (e.g., Internet IP), Transport, Presentation, Application, or even by the user [17].

The practical networking aspects of security include computer intrusion detection, traffic analysis, and network monitoring [18].

I.2.2 Objectives

Network security consists of several concepts, namely:

- **Confidentiality** : This means allowing authorized users to access sensitive and protected data. Sensitive information and data should be disclosed to authorized users only [19].
- **Integrity** : Refers to methods of ensuring that the data is real, accurate, and guarded against unauthorized user modification [18].
- **Availability** : Availability refers to the ability to access information or resources in a specified location, and the correct format [19].
- **Authenticity** : A process that ensures and confirms the identity of the user refers to authentication. The process starts when the user attempts to access information or data. The user must demonstrate access and identity rights [19].
- **Non-repudiation** : Refers to a method of guaranteeing message transmission between parties using digital signature or encryption. Proof of authentic data and data origination can be obtained by using a data hash [18].

- **Protection against Traffic analysis :** Traffic analysis involves the interception and examination of messages to deduce information from communication patterns that can be done even after messages are encrypted.

I.2.3 Network attacks

Network attacks are a collection of malicious activities that disrupt, deny, degrade, or destroy data and services in computer networks. A network attack targets the Integrity, Confidentiality, or Availability of computer network systems by exploiting the data stream on networks [17]. There are two main types of network attacks:

- **Active :** Attackers gain unauthorized access and modify data, deleting, encrypting, or otherwise harming it. Active attacks include: spoofing attack, Wormhole attack, Modification, Denial of services [17].
 1. *Spoofing:* If a malicious node fails to show its identity, the sender changes its topology [17].
 2. *Modification:* If the malicious node changes the route, the sender sends the message through the long route. This attack causes a delay in communication between sender and receiver [17].
 3. *Wormhole:* This attack can be defined as the tunneling attack. At one point, an attacker receives the packet and connects it to a malicious node on the network. So a new user supposes he has found the shortest route in the network [17].
 4. *Fabrication:* The wrong routing message is created by a malicious node. In other words, the route between devices generates the incorrect information [17].
 5. *Denial of services:* In denial of service attacks, a malicious node sends the message to other nodes and uses the network's bandwidth. The main goal of the malicious node is to make the network busy. If a non-authenticated node message comes, the receiver will not get that notice because he is busy, and the initiator has to wait for five warnings for the receiver answer [17].
- **Passive :** Attackers gain access to a network and monitor or steal sensitive information without changing the data, leaving it intact. Passive attacks include traffic analysis, Eavesdropping, and Monitoring.
 1. *Traffic analysis:* An attacker tries to perceive the communication channel between the sender and receiver in a traffic analysis attack. An attacker can determine the quantity of data traveling between the sender and receiver. The traffic analysis makes no changes to the data [17].

2. *Eavesdropping* : This is a passive attack that took place on a mobile ad-hoc network. This attack's main goal is to extract secret or sensitive information through communication. This confidential information could be a sender's or receiver's private or public key or any other confidential data [17].
3. *Monitoring* : The attacker can read the confidential data in this attack, but he is unable to update or modify the data [17].

I.2.4 Network protection

There are many ways to infiltrate a network, so IT professionals can use many different techniques and strategies to secure one. Some of the most common types of network security solutions include:

- **Antivirus Software** : Antivirus software can be installed on all network devices to scan them for malicious programs. It should be updated regularly to fix any issues or vulnerabilities.
- **Encryption** : Encryption is a good way to protect data when transmitted via a network or a distributed computer system. The protection provided by encryption is determined by the encryption method used, its implementation, and the administrative rules governing its use. Combining encryption technology with network access control mechanisms in a network security center can meet additional security requirements such as user identity, access authorization, and security auditing [20].
- **Firewalls** : In protecting networking and server resource against unauthorized access and malicious attacks, network firewalls are a primary defense line [21]. Firewalls are generally used on the network edge or at the private network entry point. Network firewalls are inspected for incoming and outgoing Internet traffic [21]. Firewalls may permit or block input or output traffic based on a set of rules. In order to do this, the network firewalls have a rule-based engine that sequentially checks incoming packets until a match is identified [21].
- **IDS (Intrusion Detection System)** : A software or hardware system for monitoring and analyzing events occurring in a network or computer system for determining if an attack occurred [22]. In other words, an IDS will always set the alarm when there is an attack, but it will not if there is no attack. IDSs are always presumed free of errors. However, an IDS is not error-free; it usually makes two possible types of error: wrong alarms (setting the alarm when no attack occurs) and missing attacks (not setting the alarm when there is an attack) [22].

Implementing secure networks requires understanding common vulnerabilities and threats that make the web the ideal environment for attackers.

I.3 SDN (Software-Defined Network)

I.3.1 Definition

Software-Defined Networking, SDN, is the programmable separation of control and forwarding elements of networking that enables software control of network forwarding that can be logically or physically separated from physical switches and routers [23]. Control logic is migrated to an entity called controller, which provides resources and abstractions to simplify the configuration of the forwarding devices with the programmability of the network through applications running on the controller [23]. The SDN architecture includes three layers: application layer, control layer, and infrastructure layer. The applications use the northbound API to communicate with the control layer, and the control layer uses the southward API to communicate with the data plane.

I.3.2 Comparison between SDN networks and traditional networks

Networks have traditionally been described by their physical topology or connected servers, switches, and routers. That means that once you have established your network, making modifications is expensive and time-consuming. This networking is incompatible with the concept of a "lights-out" datacenter¹ or a cloud environment that requires flexibility to meet changing workload demands. In addition, navigating hardware switches has become more complex as network sizes and requirements have grown. For organizations running highly virtual systems alongside massive networks, manually configuring individual network software switches has been extremely difficult and time-consuming. This is where SDN enters the scene [3].

Traditional networking uses a distributed approach for the control plane, protocols like ARP, STP, OSPF, EIGRP, BGP, and others function separately for each network device [3]. Although these network devices are connected, there is no centralized system that manages or summarizes the entire network [3]. The most crucial difference between traditional networking and SDN is that traditional networking is hardware-based, while SDN is often software-based. Because SDN is software-based, it is more adaptable, allowing users to better control and manage resources remotely in the control plane [3]. Switches, routers, and other physical hardware are used in traditional networks to create connections and operate the network. SDN controllers employ a northbound interface to communicate with Application Programming Interfaces (APIs) [3]. Because of this connectivity, instead of employing the protocols required for traditional networking, device developers can program the network directly. In traditional networks, all data planes and control planes are located in one physical unit, which is subsequently shared to increase traffic load, and the pressure on the CPU and memory in two processes [3]. Separation of control planes and data planes in SDN can be easily monitored and managed by the controller and network,

¹A lights out data center is a server or computer room that is physically or geographically isolated at an organization's headquarters, thereby limiting environmental fluctuations and human access.

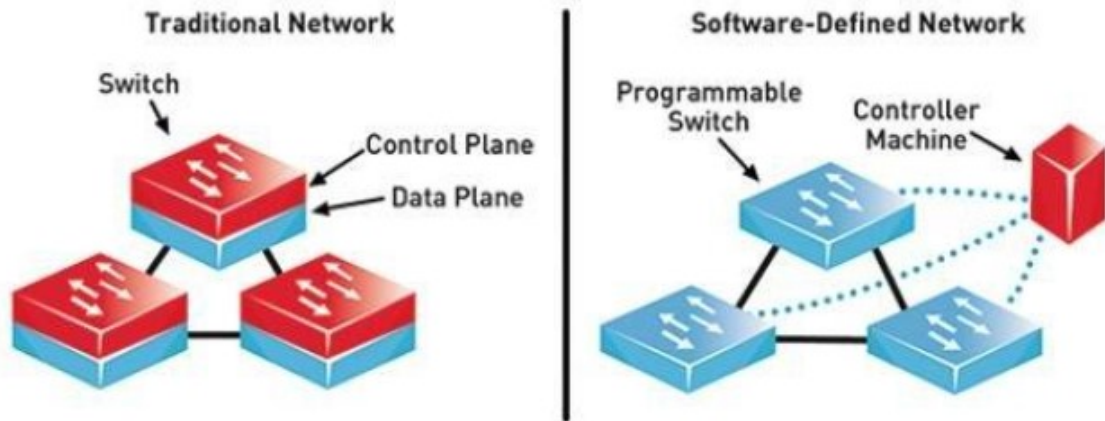


Figure I.1: The architecture of SDN Vs Traditional Network, adopted from [1]

allowing the network to better set up with less traffic load by separating these activities and having a dedicated server.

SDN is a popular alternative to traditional networks because it enables IT managers to give additional physical infrastructure services and bandwidths without investing in new hardware. The traditional network and SDN are depicted in Figure I.1. Table I.1 summarizes the key differences between traditional networking architecture and SDN architecture.

Characteristics	SDN	Traditional network
Network Control Centralized	Yes	No
Programmability	Yes	No
Flexibility of network	Yes	No
Complex Control Network	No	Yes
Performance improved	Yes	No
Configuration of Error-Prone	No	Yes
Management Enhanced	Yes	No
Configuration Efficiency	Yes	No
Easy to use and implement	Yes	No

Table I.1: The characteristics of SDN & Traditional networks architecture.

I.3.3 The architecture of the SDN

SDN Architecture shows how SDN works at various levels and assures software stability and reliability. There are three main layers in software-defined networking: Application plane, Data plane, and Control plane. As shown in Figure I.2, SDN consists of two interfaces: one between

southbound APIs (e.g., OpenFlow) and another between the API's application layer and the northbound API's control layer.

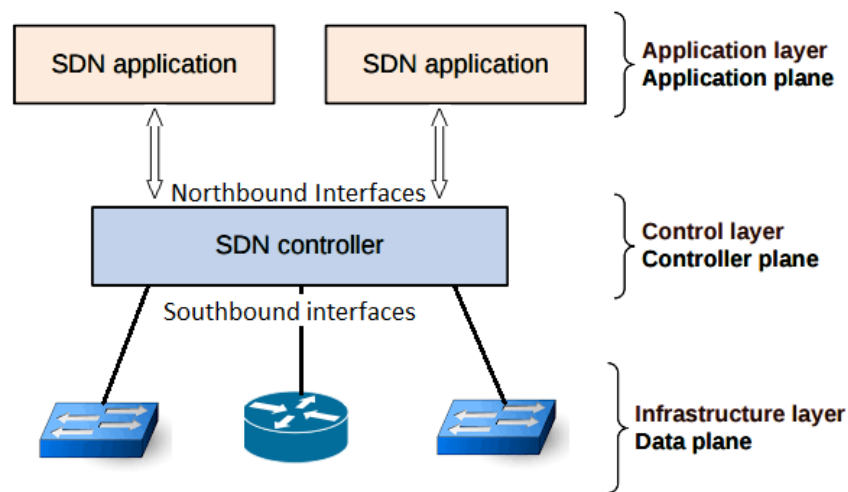


Figure I.2: Architecture of SDN, adopted from [2]

Application plane

In SDN architecture, it is the topmost layer. This layer manages all business and security applications. Essential software services managed by this layer include metering, routing, QoS, load balancer, Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS), firewall implementation, and mobility management. In addition, this layer communicates with a lower layer using the northbound application interfaces [24].

Northbound APIs

It serves as a link between the control and application planes. The NOS makes several APIs available to application developers. It helps program the network and hides the network's internal details. APIs such as FML, Procera², NetKAT³, and Frenetic⁴, among others, are widely utilized [25].

Control plane

This layer serves as a link between the application layer and the data layer. This layer contains the Network Operating System (NOS), also called the network controller, which controls the

²Procera Networks is a networking equipment company based in Fremont, California, United States, that designs and sells Network Intelligence solutions.

³NetKAT is a new network programming language that is based on a solid mathematical foundation and comes equipped with a sound and complete equational theory.

⁴Frenetic is a high-level language for programming distributed collections of network switches.

network's overall functionality. A logically centralized controller is in charge of managing the entire network and making routing, flow forwarding, and packet dropping choices via programming [26]. This layer communicates with the layer beneath it using southbound APIs such as OpenFlow and NetConf. The controller is a logically centralized and physically distributed environment that communicates with each other utilizing westbound and eastbound interfaces [26].

Southbound APIs

SDN southbound APIs are used to communicate with the SDN controller, network switches, and routers. These are protocols that allow for more efficient data plane control. Although various protocols exist, including OpenFlow, ForCES, OpFlex⁵, and Protocol-Oblivious Forwarding (POF), many organizations are working to standardize OpenFlow, which has become the de facto protocol [25].

Infrastructure plane

The data layer or data plane is another term for the infrastructure plane. It includes network components that interact with data flow, such as physical and virtual computers, like the OSI model's physical layer. This layer's primary function is to forward packets based on the controller's policies/rules assigned and developed. This layer includes physical network devices like switches, routers, and access points, as well as virtual switches like OpenvSwitch⁶, Indigo⁷, Pica8⁸, Nettle⁹, and OpenFlow [27].

I.3.4 SDN controllers

- **Beacon:** It's a java-based controller that is also modular, cross-platform, and supports both threaded and event-based functions. Beacon allows run-time modularity, which means it can start and stop programs without shutting down the main Beacon process. Beacon has a high scalability compared to other centralized SDN controllers, however it fails to address security and reliability issues [28]. Although the slicing architecture (in which each program has a constrained domain) allows it to resist privilege elevation attacks, it is vulnerable to spoofing, repudiation, and DoS attacks.
- **DISCO:** It is a distributed WAN and overlay SDN network controller. It is organized per domain, with each controller in charge of an SDN domain, and it offers a lightweight and

⁵Cisco OpFlex is a southbound protocol in a software-defined network (SDN) designed to facilitate the communications between the SDN Controller and the infrastructure (switches and routers).

⁶OpenvSwitch is an open-source implementation of a distributed virtual multilayer switch.

⁷<https://www.indigotg.com>

⁸<https://www.pica8.com>

⁹Nettle is a language for configuring routing networks.

highly controllable inter-controller channel. Agents can use this channel to share aggregated network-wide data and hence support end-to-end network services [29].

- **ONOS:** It is an experimental distributed SDN control framework based on large operator networks' performance, scalability, and availability requirements. It gives programs a global network view that is theoretically centralized and even being physically distributed across many servers [30].
- **Maestro:** It is a java-based multi-threaded controller that can manage 600K flow requests per second (rps), considerably short of the requirements of a large-scale data center (more than 10M rps) [28]. It is designed for a small domain and includes four primary applications: discovery, intradomain routing, authentication, and route flow. When it receives an incorrect OpenFlow header, it crashes and is highly vulnerable to security threats.
- **POX:** POX is an open-source OpenFlow/Software Defined Networking (SDN) controller written in Python. POX is used to create and prototype new network applications more quickly. The mininet virtual machine includes a pre-installed POX controller. The POX controller transforms OpenFlow devices into hubs, switches, load balancers, and firewalls. The POX controller makes running OpenFlow/SDN experiments simple [28].
- **NOX:** Is a first-generation network operating system based on events that can handle 30K flow requests per second [28]. This controller is designed for small organizations, domestic networks, and campus networks, but not for environments with a large volume of setup requests, such as data centers. Nicira Networks developed it to provide better performance. DoS, repudiation, and information disclosure attacks are all possible with NOX, and POX [28].
- **Floodlight:** Floodlight Controller is a software-defined networking (SDN) controller built by an open community of developers, including many from Big Switch Networks, that leverages the OpenFlow protocol to manage traffic flows in an SDN environment.
- **Ryu:** Ryu Controller is a software-defined networking framework based on components. Ryu provides well-defined APIs for software components, making it simple for developers to create new network management and control applications. Ryu can manage network devices using a variety of protocols, including OpenFlow.

I.3.5 Benefits of SDN

The benefits of SDN to organizations are numerous. A list of the major advantages is presented in the following subsections :

Programmability of the network

SDN can manage the entire network programmatically. SDN makes it easier to avoid releasing customized plans and protocols on every device in a network individually. Programmability is genuinely possible on the command plane alone, allowing the behavior of a single unit or the entire network to be changed. As a result, the controller can quickly improve traffic design functionality while also reducing network congestion [31].

Reduced price

The majority of the SDN products are free to use. Some systems, such as VMware's NSX^[10] and Microsoft's Hyper V network virtualization^[11], required only the license fee for the SDN service to be paid [31].

Enriched protection

Financing a virtual machine in a virtualized environment is a steep uphill struggle. SDN, on the other hand, provides sensitive surveillance across all devices [31].

Efficient network management

SDN allows the network manager to change the network's quality from a remote location. By changing the network characteristics based on the landing of the task in the network, easy and reliable network control is possible [31].

I.3.6 Challenges of SDN

Even though SDN has been identified as the primary solution to the increasing network's infrastructure's challenges, it is still in its infancy. Advantages such as increased functionality, lower cost, and higher efficiency have been highlighted, but different challenges must also be solved. Some of these challenges are :

Scalability

The main issue with SDN is scalability. It refers to the ability to expand to accommodate network growth. Unfortunately, the controller can become a scalability bottleneck. The introduction of distributed or peer-to-peer controller infrastructure may share the controller's communication burden [11]. However, in order to direct communications between controllers using the east and

¹⁰VMware NSX is a range of virtualization software and network security, created from vCloud Networking and Security (vCNS) VMware and Network Virtualization Platform (NVP) of Nicira.

¹¹Hyper-V Network Virtualization provides "virtual networks" (called a VM network) to virtual machines similar to how server virtualization (hypervisor) provides "virtual machines" to the operating system.

westbound APIs, an overall network view is required [11]. Aside from controller scalability, there are several other scalability issues, such as flow setup overhead and failure resilience [32].

Flexibility and performance

A fundamental issue of SDN is how to deal with high-level packet processing flows competently. In this case, flexibility and performance are the two most important variables to consider. The processing speed of a network node, taking into account both throughput and latency, is referred to as performance [11]. The SDN technique for handling new packets needs to add programmability. However, at the same time, it causes performance issues. In [33] they demonstrate that current controllers are incapable of handling a large number of flows in 10Gbps links.

Security

When compared to traditional networks, SDN's attack surface is increased by separating the control plane from the data plane [27]. According to security analysis, the SDN framework is vulnerable to a variety of security vulnerabilities, including [34]:

- Unauthorized access, such as unauthenticated application access or unauthorized controller access.
- Data leakage, such as flow rule discovery (input buffer side-channel attack) and forwarding policy discovery (packet processing timing analysis).
- Data modification, such as changing flow rules to change packets.

The following are primary possible danger sources in SDN [35]:

- Faked or fabricated traffic flows.
- Attacks on switch vulnerabilities.
- Denial-of-service attacks on control plane communications.
- Controller attacks and vulnerabilities.
- There are no methods in place to ensure that the controller and management applications are trusted.
- Attacks on administrative stations and weaknesses therein.

Figure 1.3 shows possible SDN architecture attacks :

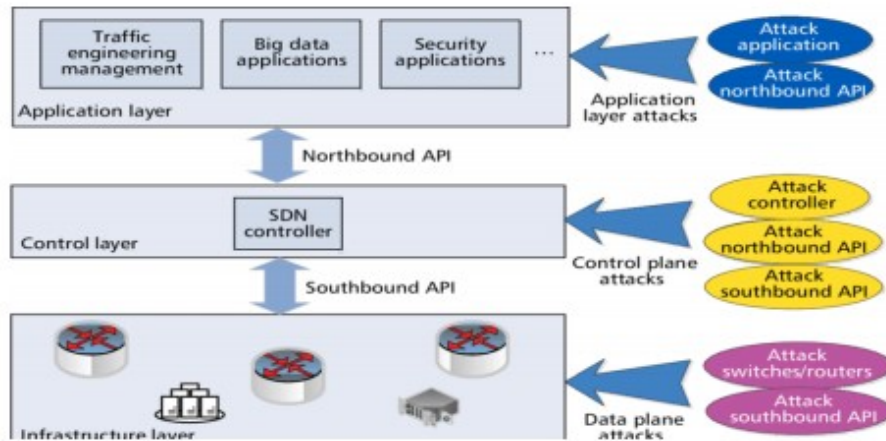


Figure I.3: Probable Attacks on SDN Architecture, adopted from [3]

I.4 OpenFlow fundamentals

The controller in SDN networks communicates with switches using open, standardized protocols. OpenFlow is the most well-known example of such a protocol.

Before getting into the technicalities of OpenFlow, it is important mentioning that there are two ways to set up physical connections to handle OpenFlow communication between switches and controllers: out-of-band and in-band [36]. Each switch has a dedicated physical connection to the controller when using out-of-band communication. Control plane information is sent through existing data plane connections between switches in in-band communication. Figure I.4 depicts these circumstances.

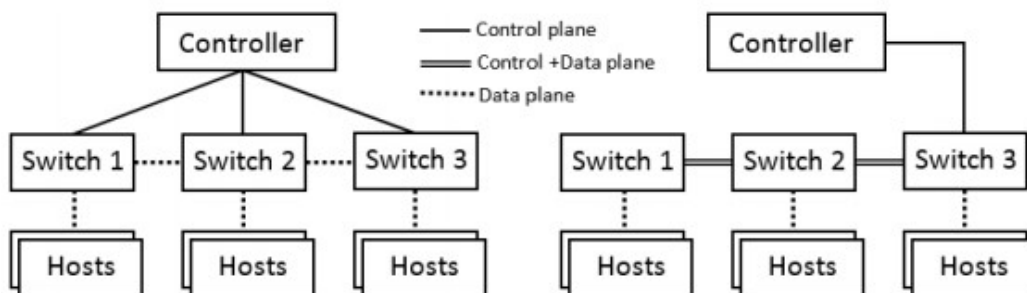


Figure I.4: Out-of-band and in-band control plane, adopted from [4]

A switch executes packet forwarding in OpenFlow by reviewing its flow table and choosing which output port to transmit. The packet header fields to match, the actions to execute on matched packets, and the related counters to update make up each entry in the flow table (also

known as a flow rule or flow entry). A switch can function at layer 2 or layer 3 of the networking stack in the context of SDN.

When a switch gets a packet that cannot be matched to any installed flow rule, the switch buffers the packet before sending an OFPT PACKET IN message to the controller to request a new flow rule. The header fields of the data packet are included in this message. The controller then sends an OFPT FLOW MOD message, which specifies the action on the data packet and the length of time the flow rule should be kept in the flow table. A timeout is a term for this period. Each flow rule has two timeout values: an idle timeout (also known as a soft timeout) that is triggered while the flow is inactive and a hard timeout that is triggered when the duration expires [37].

I.4.1 Switch Components

An OpenFlow Logical Switch is made up of one or more flow tables and a group table that handles packet lookups and forwarding, as well as one or more OpenFlow channels to an external controller (Figure I.5).

- Using the OpenFlow switch protocol, the switch communicates with the controller and manages the switch.
- The controller can add, update, and delete flow entries in flow tables both reactively (in response to packets) and proactively (using the OpenFlow switch protocol).
- Each flow table in the switch has a set of flow entries; each flow entry has match fields, counters, and a set of instructions to apply to matching packets.
- Matching begins with the first flow table and may progress to other flow tables in the pipeline.

I.4.2 Openflow ports

The network interfaces that transfer packets between OpenFlow processing and the rest are defined as OpenFlow ports. Only an output OpenFlow port on the first switch and an input OpenFlow port on the second switch can pass a packet from one OpenFlow switch to another OpenFlow switch. Several OpenFlow ports are made available for OpenFlow processing by an OpenFlow switch. Some network interfaces may be disabled for OpenFlow, and the OpenFlow switch may specify new OpenFlow ports, so the collection of OpenFlow ports may not be identical to the number of network interfaces given by the switch hardware [38].

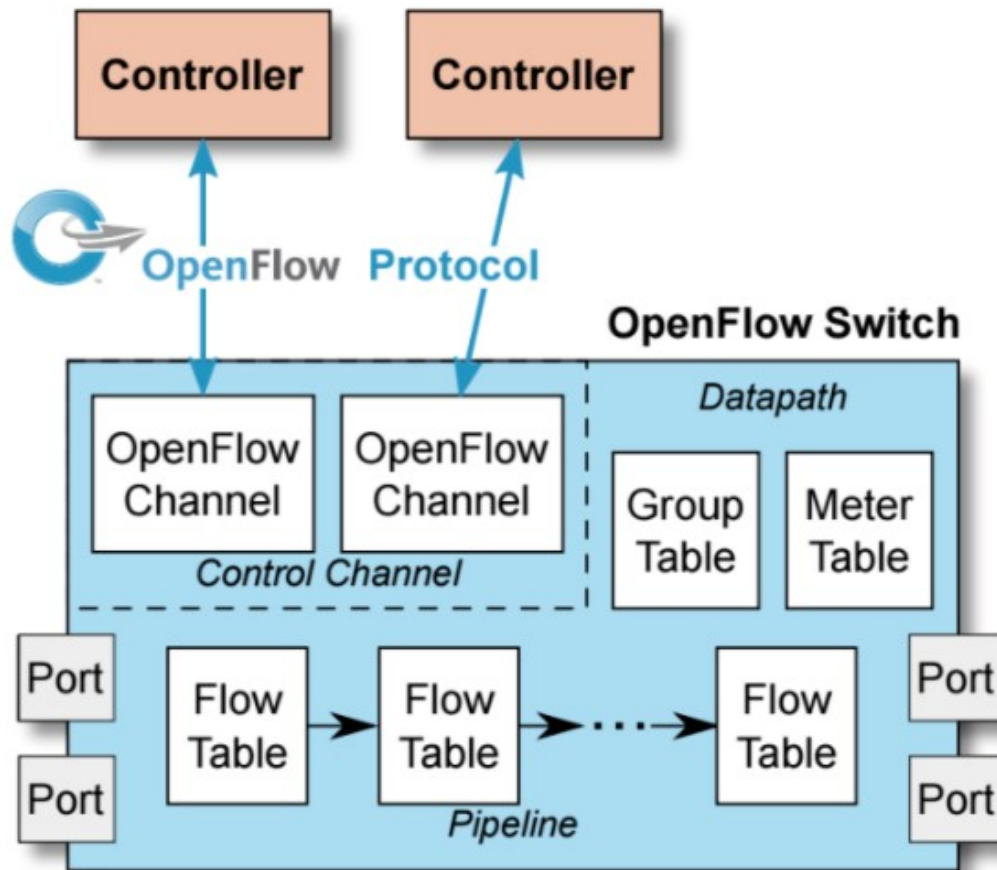


Figure I.5: Openflow switch components, adopted from [5]

The OpenFlow pipeline receives OpenFlow packets on an ingress port and processes them before forwarding them to an output port. The packet ingress port is a characteristic of the packet that represents the OpenFlow port on which the packet was received into the OpenFlow switch along the OpenFlow pipeline. When matching packets, the ingress port can be used. The output action, which determines how the packet returns to the network, can be used by the OpenFlow pipeline to decide whether to transmit the packet to an output port. **Physical ports**, **logical ports**, and **reserved ports** are the three types of OpenFlow ports that an OpenFlow switch must assist.

- **Physical ports** : The OpenFlow physical ports on the switch are switch-defined ports that match the switch's hardware interface. An OpenFlow physical port could be viewed as a virtual slice of the switch's associated hardware interface [38].
- **Logical ports** : The OpenFlow logical ports are switch-defined ports that do not immediately match to a switch's hardware interface [38]. Non-OpenFlow methods can define logical ports, higher-level abstractions (e.g., tunnels, loopback interfaces). Packet encapsulation is possible with logical ports, and they can map to a variety of physical ports. The logical port's processing is implementation-dependent and must be transparent to OpenFlow processing;

those ports must interact with OpenFlow processing the same way as OpenFlow physical ports do. A logical port packet may have an extra pipeline field called Tunnel-ID attached to it. When a packet received on a logical port is submitted to the controller, the logical and physical ports are reported. Physical ports are not affected.

- **Reserved ports** : This specification defines the OpenFlow reserved ports. In addition, they define generic forwarding activities like sending to the controller, flooding, or using non-OpenFlow methods like "normal" switch processing [38].

I.4.3 Openflow channel

Each OpenFlow Logical Switch communicates with an OpenFlow controller through the OpenFlow channel. The controller configures and manages the switch through this interface, receives events, and sends packets. A single OpenFlow channel with a single controller may be supported by the switch's Control Channel or several OpenFlow channels with multiple controllers may be supported by the switch's Control Channel. Each OpenFlow Logical Switch is connected to an OpenFlow controller through an OpenFlow channel. The controller configures and manages the switch using this interface, receives events from the switch, and sends packets. A single OpenFlow channel with a single controller may be supported by the switch's Control Channel or several OpenFlow channels with multiple controllers may be supported by the switch's Control Channel. The OpenFlow channel is normally encrypted with TLS, but it can also be performed via TCP [38].

I.5 Denial of Service

On Thursday, 6th August 2009, one of the most widely followed social networking and micro-blogging online services, Twitter¹², was brought down for several hours, silencing its millions of Tweeters [39]. The incident was a result of what it described as an "ongoing" denial-of-service attack. The first official word that came in was a rather terse statement: 'Site is down – We are determining the cause and will provide an update shortly. However, its co-founder Biz Stone soon updated this message: 'On this otherwise happy Thursday morning, Twitter is the target of a denial of service attack.'

I.5.1 Definition

A Denial of Service (DoS) attack is one single attacker against one target. This kind of attack denies the access of other legitimate users to shared services or resources. DoS attacks accomplish this by flooding the target with traffic or sending it information that triggers a crash so the server cannot distinguish between valid and non-valid requests. In both instances, the DoS attack

¹² <https://twitter.com>

deprives legitimate users (i.e., employees, members, or account holders) of the service or resource they expected.

There are two general methods of DoS attacks: flooding services or crashing services. Flood attacks occur when the system receives too much traffic to buffer, causing them to slow down and eventually stop. Other DoS attacks exploit vulnerabilities that cause the target system or service to crash. In these attacks, input is sent that takes advantage of bugs in the target that subsequently crash or severely destabilize the system to be accessed or used.

I.5.2 Distributed denial of service (DDoS) Attacks

Unlike simple DoS attacks, DDoS attacks take advantage of many compromised hosts and aim to exhaust network resources as fast as possible. Once the network is disrupted, it cannot provide any services to legitimate users [39]. The strategy of DDoS consists of recruiting multiple agent (slave) machines. These machines are usually used to send the attack packets to flood the target. The attacker exploits the vulnerable machines automatically using special tools for this, and then it will use these infected machines to recruit new agents in the future.

I.5.3 Botnets

A Bot, which comes from the term 'robot,' is an application that can perform a specific task more quickly than a human being. When a large number of bots spread to multiple computers and connect via the Public Internet, they form a botnet, which is a bots network [40]. Three principal elements - bots, C&C¹³ servers, and botmasters - make a botnet [40]. A bot is designed to infect targets (e.g., computers or mobile devices) and to make them part of a botnet under a person known as the botmaster without their owner's knowledge. The botmaster commands and controls the whole botnet through the Internet using C&C servers [41]. Thus, these targets are being controlled by botmasters, and malicious activities take place. An examination of the various types of malicious activity committed by botnets shows that they pose a dangerous threat not only to computers and the Internet but also to other attacks and acts (e.g., DDoS) [42] as infrastructure.

I.5.4 Taxonomy of DDoS attacks

DDoS attacks have shown several variations in their attacking methods in past years and are still getting experimented with several possibilities [43]. that was the reason behind the variation of the proposed survey about the DDoS taxonomies. Mirkovic and Reiher [44] presented taxonomies for DDoS classification, attacks, and possible defense mechanisms. The attacks were categorized as: automation, vulnerability, source address validity, attack rate dynamics, characterization, the

¹³C&C : Command and Control server

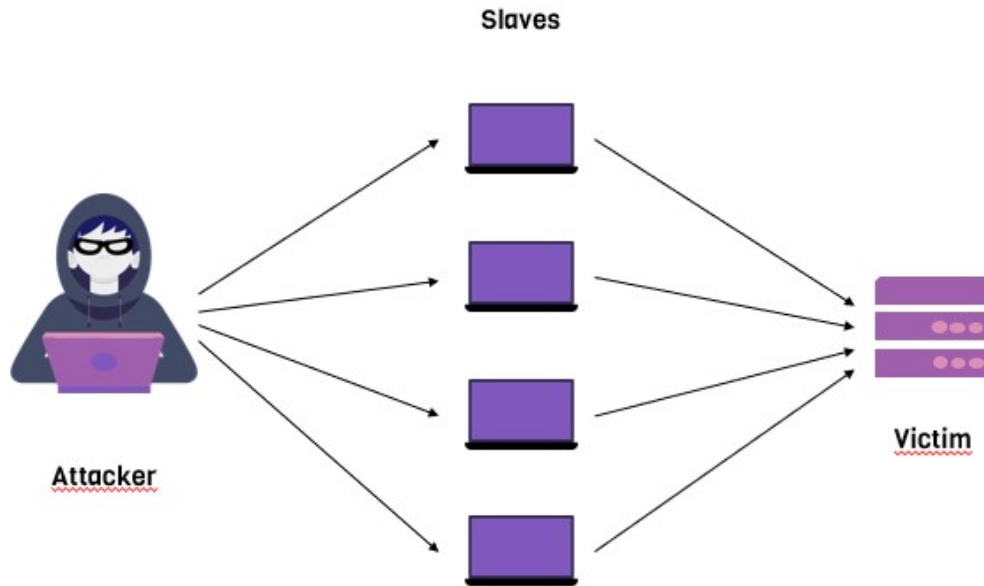


Figure I.6: DDoS attack scenario

persistence of agents, victim, and impact on the victim. Asosheh and Ramezani [45] proposed taxonomy based on known potential attacks and categorized attacks based on eight features: architecture, degree of automation, impact, vulnerability, attack rate dynamics, scanning strategy, propagation strategy, and packet content.

Bhardwaj et al. [46] proposed the taxonomy for the various potential DDoS attacks. These are four categories: degree of automation, vulnerability, attack rate dynamics, and attack impact.

It is highly recommended to know the classified nature of attacks to make an effective defense. That is why we choose the classification of [47], which is categorized into two classes, reflection-based, and exploitation-based attacks.

- **reflection-based DDoS** : Reflection occurs when an attacker forges the source address of request packets, pretending to be the victim. This technique hides the real IP address of the attacker from both the victim's system and the abused server. Servers are unable to distinguish legitimate from spoofed requests. Therefore, they reply directly to the victim.

Specific attacks can be carried out using either TCP or UDP like DNS, LDAP, NET-BIOS¹⁴, and SNMP [47]. In addition, these attacks can be carried out through application

¹⁴NETBIOS is a network architecture co-developed by IBM and Sytek in the early 1980s. It is not a network protocol, but a naming system and a software interface that allows sessions to be established between different computers.

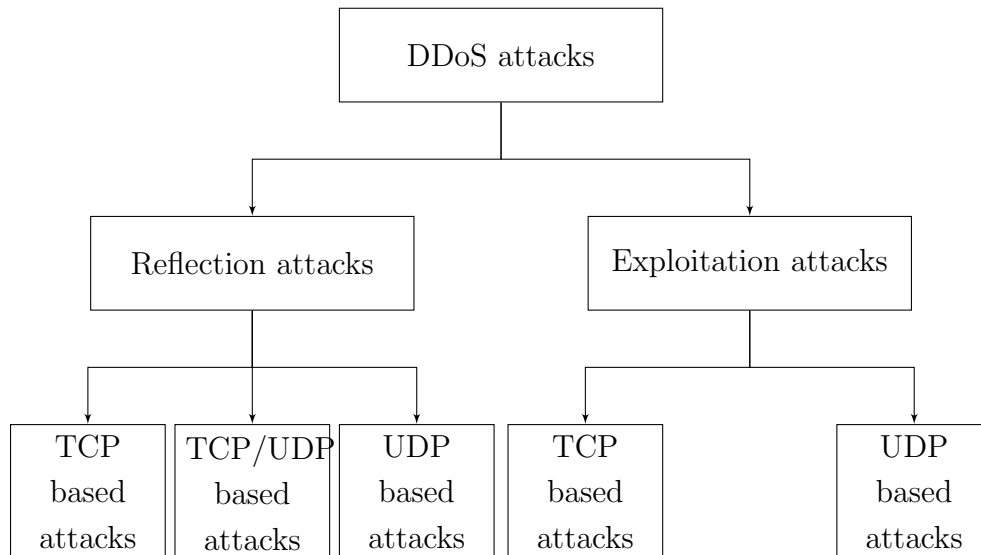


Figure I.7: DDoS attacks Taxonomy.

layer protocols using transport layer protocols, i.e., Transmission control protocol (TCP), User datagram protocol (UDP), or through a combination of both. In this category, TCP-based attacks include MSSQL¹⁵, SSDP while UDP-based attacks include CharGEN¹⁶, NTP, and TFTP.

- **Exploitation-based attacks** : These attacks can also be carried out through application layer protocols using transport layer protocols, e.g., TCP and UDP. TCP based exploitation attacks include SYN flood, and UDP based attacks include UDP flood and UDP Lag [47]

1. **TCP flood attack:** TCP flood attacks are DDoS attacks in which an attacker sends a large number of packets to a victim system in an attempt to deplete its resources or use bandwidth [48]. Before data transmission, the client and server connections should be established in a TCP connection. TCP three-way handshake is what it is called [6]. In this attack, The client must send a SYN message to the server, which will be confirmed by the server sending a SYN-ACK message to the client, who must then send an ACK message to the server, and the connection will be opened [6]. When the attacker uses a fake IP address to transmit repeated SYN packets to a random port on the targeted server, the standard TCP three-way handshake will change into a TCP SYN flood [49], as seen in Fig. I.8. The server may face issues such as trouble closing the connection (connection remains open) and receiving a high number of SYN packets with no response to legitimate clients, which can cause the server to crash [6].

¹⁵MSSQL is a suite of database software published by Microsoft. It includes a relational database engine, which stores data in tables, columns and rows.

¹⁶The Character Generator Protocol (CharGEN), was designed to be used in debugging and measurement tasks, which are used check the status of network connections, whether the buffer is working properly, and possible

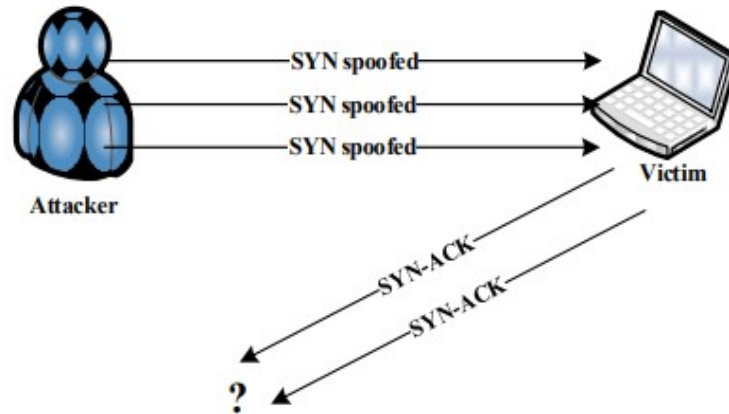


Figure I.8: TCP SYN flood Scenario, adopted from [6]

2. **UDP flood attack:** In this attack, the attacker tries to send UDP packets to random ports on the target device while remaining anonymous [43]. Therefore, the target device must check each port for an application to listen to, but it responds with an ICMP destination unreachable packet because there is no one. Due to busy waiting, this entire operation renders the target device unavailable [43].
3. **HTTP flood attack:** The cybercriminal uses real HTTP GET or POST requests to launch a DDoS attack in this attack [44]. These attacks do not involve spoofing or reflection techniques. Therefore they take less bandwidth to reach the targeted server than other attacks [43].
4. **ICMP flood attack:** ICMP is generally in charge of sending error messages to the source to alert them of any network or destination failures. An example case could be when the gateway cannot buffer data or when a packet is not reachable to its destination. The ping function in ICMP sends out an echo request, which is then responded with an echo reply. If you do not get a response, that means the other host is not online or does not have ping capability [43]. In that case, echo requests continue to be issued without waiting for an echo response, flooding the network and wasting bandwidth [50]. A smurf attack is another name for it.

I.6 Conclusion

As an emerging network technology, SDN has a bright future, with security being one of the most urgent issues. Unfortunately, DDoS attacks have become more sophisticated over time, and each year, large-scale attacks with increasing frequency and size are launched against huge companies, data centers, and other businesses [9].

electronic limitations.

In this chapter, we presented notions of network security, SDN networks, and DDoS attacks. A definition of the SDN network has been provided, its architecture, controllers, benefits, and challenges. We also made a brief comparison between traditional networks and SDN networks. Finally, we discussed DDoS attacks and their effects, botnets, and taxonomy, containing several DDoS attack types.

Chapter II

DDoS attacks in SDN networks, and ML approach

II.1 Introduction

SDN's key benefit is its ability to decouple the data and controller planes, providing developers more flexibility and making application maintenance easier in these situations. In addition, it was able to separate the network infrastructure from its applications to decouple the control plane and be logically centralized. As a result, the software can build services now implemented in hardware (IDS, Firewalls, and Routers). This centralization and decoupling of SDN are possible because the network controller and switches have well-defined programmable interfaces via APIs that allow them to communicate with one another.

Despite the potential for innovation introduced by SDN, it introduces some new challenges to network security. The core qualities of a secure communication network, such as secrecy, integrity, information availability, authentication, and nonrepudiation, can be compromised by these issues. A significant number of packets are transmitted to a connected host or set of servers. The gathering of authentic and counterfeit DDoS packets can exhaust the controller's processing capacity. This will render the controller unreachable to legitimate newcomer packets, potentially destroying the SDN architecture. Because intruders might be spread and situated on multiple switches, DDoS attacks can be challenging to detect. Because switches cannot detect attacks completely, completing the detection process may not improve the detection process. DDoS assaults, such as controller-switch communication flooding, can have a significant impact on an SDN network.

Devices such as controllers, sensors, and communication substrates in critical infrastructures might exacerbate such issues in IoT scenarios [51]. However, in recent years, some solutions to SDN security concerns have been offered in the literature [51]. Machine learning algorithms are one way that can be utilized to address these challenges. In this chapter, we will present a description of the DDoS attack defensive mechanisms in SDN networks.

The goal of SDN is to provide open interfaces for the development of software that can control the connectivity provided by a set of network resources and the flow of network traffic through them, as well as possible traffic inspection and modification in the network.

As shown in Figure II.1, the adopted SDN architecture consists of three levels. The adoption of this layered architecture offers many advantages but also various security disadvantages. In short, security problems with SDN can be found at three levels: the communication level between different architectural elements, the components level, and the audit and logging level.

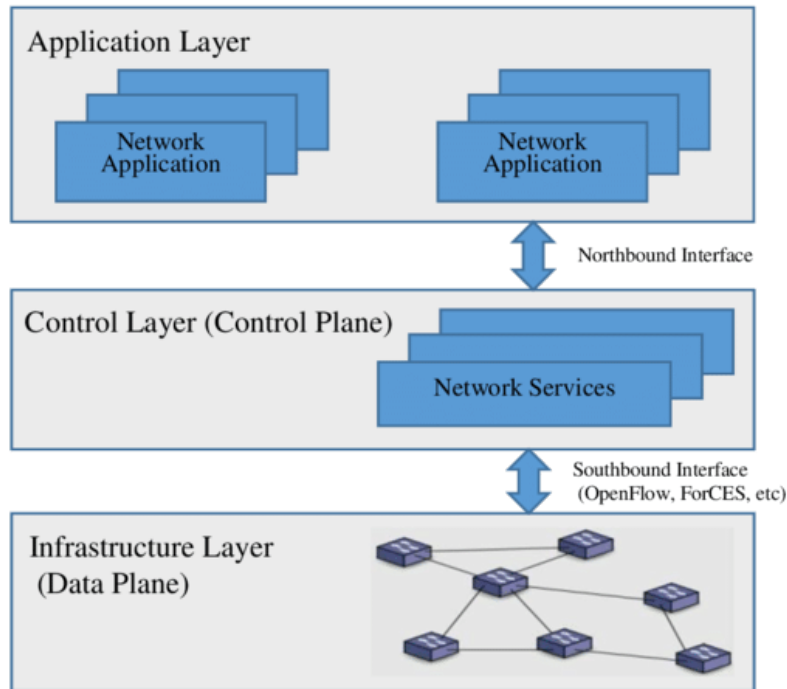


Figure II.1: SDN architecture and components, adopted from [7]

II.2 Machine learning approach

We apply machine learning approaches to analyze system performance and detect odd occurrences that are not compatible with typical network behavior. Abnormal actions are recognized by mathematical models constructed using machine learning techniques, especially in network systems where high-density data flows and rapidly applying preventive policies. The characteristics of the machine learning approaches employed in this work are briefly described in this section.

II.2.1 Introduction to machine learning

Machine learning is a technique of developing computer algorithms that can emulate human intelligence. It incorporates concepts from artificial intelligence, probability and statistics, computer science, information theory, psychology, and control theory, among other areas [52].

Machine learning (ML) is a term that refers to intelligent techniques for optimizing performance criteria by learning from example data or prior experience(s). More precisely, ML algorithms use mathematical techniques to create behavior models from large data sets [53]. Learning without being explicitly programmed is also possible with machine learning. These models are then utilized to make future predictions based on the newly entered data. Artificial intelligence, optimization theory, information theory, and cognitive science are just a few of the science and engineering

fields that have influenced machine learning [54].

II.2.2 Types of machine learning algorithms

There are four types of machine learning algorithms: supervised, unsupervised, semi-supervised, and reinforcement learning algorithms.

- **Supervised Learning:** When particular targets are defined to be reached from a collection of inputs, supervised learning is used. The data is first labeled, then trained with labeled data (with inputs and desired outputs) for this sort of learning. It attempts to automatically identify rules from accessible datasets, create various classes, and forecast whether items (objects, individuals, and criteria) belong to a specific class [54].
- **Unsupervised Learning:** The environment mainly provides inputs in unsupervised learning, with no desired targets. It can analyze similarities among unlabeled data and classify them into distinct groups without requiring labeled data [54].
- **Semi-supervised Learning:** In the preceding two types, either all of the observations in the dataset have no labels or all of the observations have labels. Semi-supervised learning is more in the middle. In many practical instances, the cost of labeling is relatively high because it necessitates qualified human experts. As a result, semi-supervised algorithms are the best options for model development when labels are absent in most observations but present in a few [54].
- **Reinforcement Learning:** No explicit outcomes are established in Reinforcement Learning (RL), and the agent learns via the feedback after interacting with the environment. It takes some acts and makes decisions based on the reward it receives. In addition, human and animal learning patterns influence it. Such characteristics make it an appealing technique is highly dynamic robotics applications where the system learns to do specific tasks without explicit programming. It is also crucial to pick the proper reward function because the agent's success or failure is determined by the overall reward accumulated [55]. Reinforcement learning techniques are generally used in the following scenario [54]:
 - When historical data and past examples are unavailable for model training
 - The overall goal is known, and the environment can be detected to maximize both short and long-term results.
 - when the precise right and wrong values for a particular scenario are unknown a priori

II.3 Supervised machine learning

II.3.1 k-Nearest Neighbor (k-NN)

One of the most widely used machine learning algorithms is k-NN. It is a supervised, non-parametric, distance-based approach that was first introduced in 1951 [56]. By using a distance function, this algorithm calculates the dataset's similarities. The classification of the test data is based on the majority votes of its k-nearest neighbors.

X and Y pairs make up a training set. Let $X = x_1, x_2, \dots, x_n$ represent the n-dimension feature set's training data, and $Y = y_1, y_2, \dots, y_n$ represent the target labels. The following is a prediction for a test data x used as input to the k-NN model [57]:

- To determine similarity in the training data, a distance function such as a Euclidean one is utilized. The distance between two places labeled a and b with Cartesian coordinates (a_1, a_2) and (b_1, b_2) is determined as shown in Equation :

$$d(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$

- The majority votes of its k-nearest neighbors are used to decide the label of test data x.

- **Advantages :** [54]

1. a straightforward (only two parameters required to implement KNN, i.e., the value of K and the distance function (e.g., Euclidean or Manhattan, etc.) algorithm that can be used as an initial assessment of simple classification in small networks.
2. The performance of the algorithm is degraded in large datasets and bigger networks. This is since the cost of the distance calculation between new points and each existing point is huge. It is also sensitive to the noisy data.
3. KNN does not work well with high-dimensional data. This is because of the fact, high computational cost, for distance calculation in each dimension, is associated with a large number of dimensions.
4. KNN is a high-speed algorithm and does not require any training period (e.g., SVM, Linear Regression etc.). It stores the training dataset and learns from it only at the time of making real-time predictions.

- **Disadvantages :** [58]

1. It is computationally costly.
2. It is susceptible to irrelevant features.
3. It is a lazy algorithm (it takes more time to run).
4. It needs huge memory to store all the training examples.

II.3.2 Decision Tree

For regression and classification of actual situations, the decision tree machine learning algorithm is applied. This model is based on the structure of a tree. The tree's base, on the other hand, is at the very top. The decision tree is also gradually developed, with the branches created using objective rules based on the dataset's attributes [59].

The steps listed below can be used to generate a decision tree [60]:

1. The entire dataset is split into two: training and test sets.
2. The training set is used as an input to the tree's root.
3. Using the information theory presented in the equation, the root is identified

$$Entropy(P) = - \sum_{i=1}^N p_i \log(p_i)$$

4. The prone process is operated.
5. The listed methods are repeated until all nodes have become leaf nodes.

where p denotes the dataset's probability distribution. Other hyper-parameters to achieve an efficient decision tree include minimum leaf size, parent size, and maximum splits.

- **Strengths :** [58]

1. Decision Tree is a self-explanatory tool since it has a simple schematically representation that can even be followed by the non-professionals.
2. Decision Tree can easily be converted to a set of rules which are often comprehensible for the reader.
3. Decision Tree is a non-parametric tool. Therefore it does not require any functional form specification.
4. Decision Tree can easily handle outliers and missing values.

- **Limitations :** [58]

1. Decision Tree can be computationally expensive.
2. Decision Tree can easily overfit the data, but in practice, there are several tools to avoid overfittings, such as post-prune and pre-prune.
3. In practice, the decision tree is widely used for classification and less appropriate for estimation tasks in regression.

II.3.3 Support Vector Machine

In terms of classification and regression analysis, the SVM method is one of the most efficient machine learning algorithms. When the space is divided into two or more classes, SVM determines support vectors that can do so. Whenever possible, a big margin is used, and the support vectors are the data points that make up this boundary [61]. As part of SVM, a non-linear kernel is utilized to split the data. The SVM searches for support vectors, weights, and biases to achieve this goal.

- **Advantages :**

1. It has the ability to work with binary as well as with multi-class environments [54].
2. Works well with unstructured and semi structured data such as text, images, and trees [54].
3. SVM algorithm is not suitable for large data sets [54].
4. Avoid the optimum local issue;
5. Strong generalization capability;
6. Can process small samples;
7. Robustness.

- **Disadvantages :**

1. Classification has limitations;
2. Time consuming;
3. Not sensitive to missing data under larger-size samples
4. the difficulty to interpret unless the features are interpretable. It can be computationally expensive, and it needs a good kernel function. [58]
5. Its lack of transparency in results because it is a non-parametric method. [58]

II.3.4 Random Forest

LEO Breiman and Adele Cutler created the random forest. These classifiers combine various decision trees to predict new unlabeled data; each decision tree is present in the forest, and its quality is determined by the number of trees present [62]. Random attributes are applied to each tree, each number of trees corresponds to a single forest, and each forest serves as a predation class for new unlabeled data. For each tree, this approach uses a random feature selection method. The outputs are then classified and predicted using a collection learning algorithm in light of a specific number of trees. Several classification trees are formed using this method, and each free tree is constructed using a different component of the overall dataset. Following the classification

of each tree into an unmarked class, another question will be added under each tree vote in favor of choice.

- **Advantages :**

1. Avoid over-fitting;
2. Can handle high dimensional, continuous and discrete data;
3. Strong anti-noise ability, avoid over-fitting, small generalization error;
4. Fast training speed and high accuracy

- **Disadvantages :**

1. Over-fitting especially in noisy classification or regression problems;
2. Black box model, weak interpretability;
3. Time-consuming under large gain.

II.3.5 Logistic Regression

For binary classification, a common Machine Learning approach is logistic regression. Logistic regression can be used to solve problems that are binary, ordinal, or multinomial. The binary logistical regression is used in cases when only one of two possible outcomes for a dependent variable can be observed: awardee or non-awardee (0 or 1) [63].

II.3.6 Naive Bayes

The Bayes theorem is used to create a collection of classification algorithms known as Naive Bayes classifiers. Naive Bayes is not a single algorithm but rather a group of algorithms that share a common principle [64].

- **Advantages :** [54]

- Used in the binary and multi-class environment.
- Effective in anomaly and intrusion detection problems.
- It works best when used with discrete data, and it can fall into the wrong prediction if continuous data is used.

- **Disadvantages :** [65]

- The Naïve Bayes classifier requires large number of records to get good results.
- May be less accurate than other classifiers applied on some datasets.

II.4 Dataset

Dataset is a collection of related observations organized and formatted for a particular purpose [66]. Datasets have played a foundational role in the advancement of machine learning research. They form the basis for the models we design and deploy, as well as our primary medium for benchmarking and evaluation [67].

II.4.1 Types of dataset

- **Real dataset** : It represents the data generated from actual world events, which means data from a production system, vendor, public records, or any other dataset that otherwise contains operational data. For example, a dataset that is a ten-year-old backup of an existing system and contains data about real individuals, matters, or cases, would be real data [68].
- **Synthetic dataset** : Synthetic data is generated from real data by using the underlying statistical properties of the real data to produce synthetic datasets which exhibit these same statistical properties [69]. An excellent synthetic dataset should replace sensitive values and provide more robust guarantees of privacy and anonymity. Synthetic data can be used in two ways:
 - To increase the size of a dataset, for times when a dataset is unbalanced due to the limited occurrence of an event.
 - To generate a full synthetic dataset representing the original dataset, for times when data is unavailable due to its sensitive nature.

II.5 Data Preprocessing

This phase is essential to achieve data quality. So, data reduction is made by selecting the most important attributes without losing quality [65].

II.5.1 Feature Selection

Before using a learning algorithm, feature selection is frequently an essential data processing step. For example, machine learning algorithms frequently improve their performance by removing irrelevant and duplicated data [70]. Feature selection is the process by which a data scientist selects a subset of relevant characteristics to employ in machine learning model development, either automatically or manually. It is, in fact, one of the essential principles in machine learning that has a significant impact on the performance of your models since it is the key to developing dependable machine learning models. The procedure will choose the optimal subset of attributes from a pool of most important features and have a high contribution at the prediction time.

II.5.2 Feature Selection Procedure

In the FS method, the full feature set is considered for classification at first. The FS methods are then used to choose the features. For model evaluation, the selected features are combined with a classification method. The following are the basic steps in the FS process [8]:

- Generating the subset of features.
- Evaluating the generated feature set.
- Setting a termination criterion.
- Validating the results for the specified subset of features.

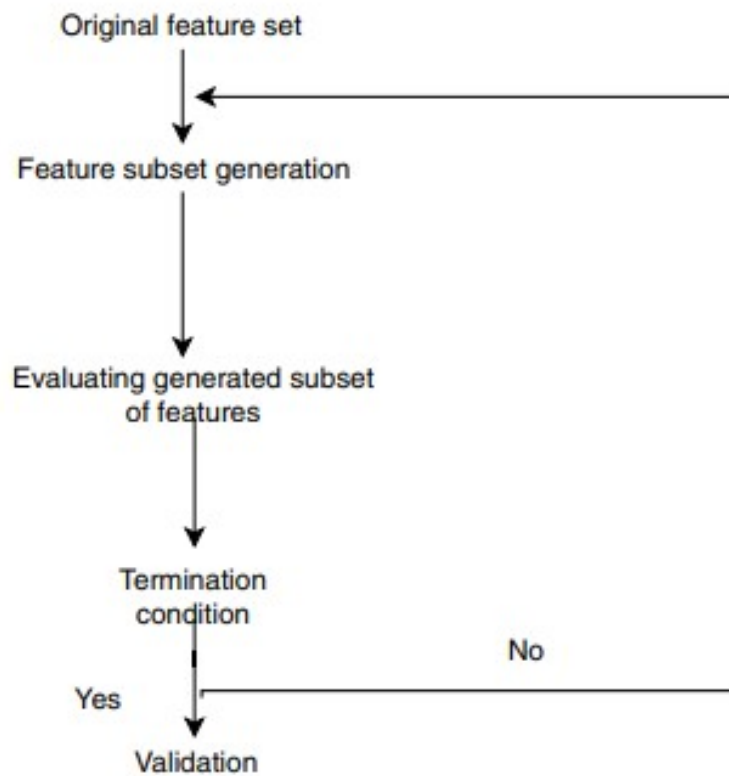


Figure II.2: Feature selection process, adopted from [8]

II.5.3 Feature selection methods

Feature selection methods can be classified into three groups in general:

- **Filter Methods:** Rather than utilizing a machine learning technique, rely on the properties of the features. It is ideal for a quick "screen and remove" of unwanted features. The filter method eliminates aspects that do not add significantly to data analysis [71].

Figure II.3 depicts a schematic of a filter-based technique. This method creates feature sets with a larger number of features, and in certain cases, the whole feature set is selected.

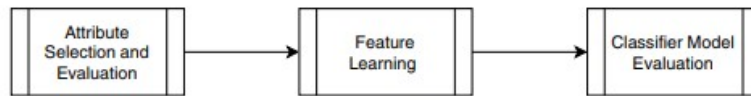


Figure II.3: The filter method, adopted from [8]

- Embedded methods:** The selection of a group of features is treated as a search problem, with the optimal feature subset chosen using a predictive machine learning method. These methods, in essence, train a new model for each feature subset, which is computationally expensive. However, provide the optimum feature subset for a specific machine learning algorithm. As illustrated in Fig. II.4, the embedded feature selection method is a combination of wrapper and filter-based selection methods. It makes use of the FS approach, either implicitly or explicitly, to increase the classifier's performance [8].

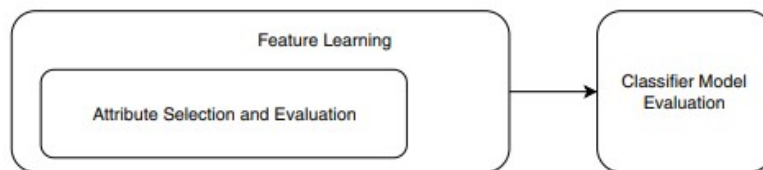


Figure II.4: The embedded method, adopted from [8]

- Wrapper methods:** The wrapper-based FS can be divided into two parts: search and evaluation [72]: The search process is concerned with parameter initialization, which is required for feature evaluation using an evaluation function. The wrapper-based FS algorithm interacts with the classifier to infer the importance of features for feature selection because it is dependent on the classification algorithm. In comparison to filter and embedded based FS techniques, the wrapper based FS method is slower. For searching the features, it uses forward selection and backward elimination [72].

II.6 Classifier Evaluation Measures

There are several metrics for evaluating classifier performance. There is no single metric that can tell us everything about the classifier's performance [58]. Some specific measures were used to assess the classifier's performance.

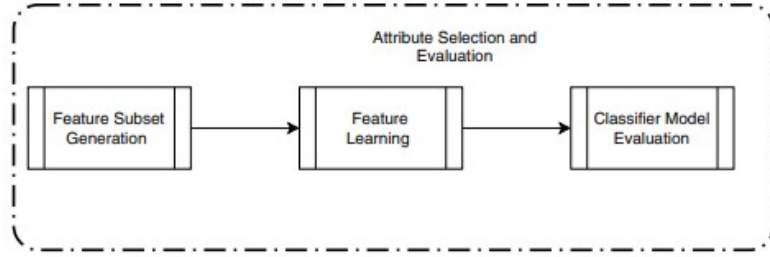


Figure II.5: The wrapper method, adopted from [8]

II.6.1 Accuracy

The accuracy of an algorithm is a measure of how well it classifies unseen occurrences, and it can be calculated using the formula below:

$$Accuracy = \frac{\text{Number of correctly classified instances}}{\text{Total number of instances}} \times 100$$

II.6.2 Confusion matrix

Accuracy is not the only way to assess performance; sometimes, we need a complete picture of the classifier's performance. One such detailed table is the Confusion Matrix. The matrix provided in table II.1 is commonly used to evaluate performance: The following rules can be extracted from

	Predicted	
	Positive	Negative
Positive	TP	FN
Negative	FP	TN

Table II.1: Confusion matrix

the table above:

- **Accuracy** - The percentage of occurrences adequately classified by a classifier :

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}$$

- **Precision** - DDoS attack precision is calculated by dividing the total number of expected true and false DDoS attacks by the total number of predicted true and false DDoS attacks.

$$Precision = \frac{TP}{TP + FP}$$

- **Recall** - is calculated by dividing the total number of predicted DDoS attacks by the total number of actual DDoS attacks.

$$Recall = \frac{TP}{TP + FN}$$

- **Error rate -**

$$Errorrate = 1 - Accuracy = \frac{FN + FP}{TN + FN + FP + TP}$$

II.7 SDN Security problems

II.7.1 Communication Level

1. At the northbound communication level, the key security problems are:

- Failure to trust and weak authentication between the applications and the controller can spoof northbound API messages.
- Inappropriate authorization on the applications can trigger inappropriate or malicious access.

2. At the southbound communication level:

- The lack of encryption between the controller and the switches allows eavesdropping and spoofed southbound communications.
- The lack of trust and weak authentication at this level can lead to man-in-the-middle attacks and spoofing attacks, which makes it easier for an attacker to eavesdrop on the flow to see the flow in use and the network that allows traffic across that flow
- Improper authorization may lead to unauthorized access. Suppose a user requests to trigger the controller to create a route and cause a packet to traverse the routed service. In that case, we must ensure that the user is authorized to do so and that the environment can resolve the activity.
- Southbound communication is impacted by flow rule attacks, primarily in in-band deployments.

II.7.2 Each Component level

Every component of the SDN architecture must be protected.

1. The application component

- If the applications are not verified to be from trusted sources, we risk having vulnerable or malicious applications that create major vulnerabilities and, as a result, compromise the whole SDN network.
- The API itself may be vulnerable. If the attacker can use the vulnerable API, the attacker can control the SDN network through the controller. If the controller lacks any form of security of the northbound API, the attacker may create his SDN strategy to gain control of the SDN environment.

- Inconsistent flow rules may result from a lack of application isolation. Flow isolation uses tags to distinguish packets belonging to different policies, allowing for consistent network updates and, as a result, consistent flow rules.

2. The Controller component

- The controller is vulnerable to Denial of Service attacks, compromising the entire network infrastructure. Spoofed packets used in the various types of messages used by OpenFlow or OpFlex (handshake, hello, and others), as well as bad logic in the controller's code, could cause legitimate network devices to be shunned and potentially cause Denial of Service (DoS) problems.
- The attacker's implementation of rogue controllers could result in entries in the flow tables of network elements. As a result, the attacker would have complete command of the network.
- Controller Hijacking occurs when a hacker takes over the controller and gains complete control of the SDN network, flows, and defined policies.

3. The Switch component

- On the flow table level, the possibility of a DoS attack on the switches exists; the attacker can flood the switch with massive flow rules.
- The possibility of various attacks on the agent installed on a switch (e.g., buffer overflow, denial of service, etc.).
- Network instability and unavailability can be caused by inconsistencies in flow rules.
- When the switch's Listener Mode is enabled, it allows a connection from the controller to the switch with no built-in authentication or access control.

II.7.3 Logging and audit level

The audit and compliance component has been identified as a major security issue in SDN. The lack of logs and audit trails can result in a loss of control over the SDN network, with no monitoring or checking for unauthorized changes or malicious attempts by administrators or attackers.

II.8 Defeating DDoS attacks in SDN based Networks

DDoS attacks are growing in size, frequency, severity, and sophistication in the traditional network. It means that DDoS attacks with current IDS are limited to solving the problem. Attackers have developed more advanced methods for circumventing existing defenses. SDN has certain features, instead, that have advantages to some degree in overcoming DDoS attacks.

- **Separation of the Control Plane and Data Plane:** The experimentation in traditional networks is complicated. Moreover, because the control logic is integrated into devices, it is too difficult to examine newly used algorithms in traditional networks. Every device should be individually updated. SDN, however, separates two planes below and simplifies the testing of comprehensive mechanisms. SDN has large dynamic configuration functionality, which enables experimental environments.
- **Global View Network:** The controller monitors network traffic and has a global view of the network. Centralizing of SDN controller makes it easy to isolate the compromised host from the legitimate host using information obtained by requesting the end hosts [25].
- **Traffic Analysis based on Software:** To monitor and configure network devices, various application utilities run in the SDN application plane. Many software and algorithms are available to analyze the network traffic, which reduces the burden of switches to parsing the traffic [25].
- **Programmability of Network:** The application plane contains applications that program the controller and further control the network's behavior. The programmability of the SDN network makes it more flexible because more intelligence can be deployed at any time [73]. Also, because the network is programmable, incoming traffic can be analyzed to identify malicious traffic or hosts and maintain network performance.
- **Dynamic Network Policy Update:** The immediate response in DDoS attack mitigation is the dynamic adjustment of flow rules on OpenFlow switches. Based on the analysis of the traffic, new innovative algorithms to block the traffic can be propagated instantly [74]. Adding a new rule to every device is brutal in a traditional network, but updating switches dynamically is simple with SDN.

II.9 Types of DDoS attacks in SDN

The network SDN architecture is a new concept that decouples the data plane from the control plane to improve network manageability and security. Separation of Control plane from the data plane, global view of the network, the programmability of the network, traffic analysis based on the software, and dynamic updating of network policies are some inherent characteristics of the SDN to enhance the security of the network [25]. Nonetheless, various attacks infect the data plane, control plane, and plane-to-plane connections. These attacks are destroying the SDN architecture and making it a problematic network architecture.

In SDN architecture, Figure II.6 demonstrates which plane is vulnerable to a specific attack. The following are some examples of such attacks.

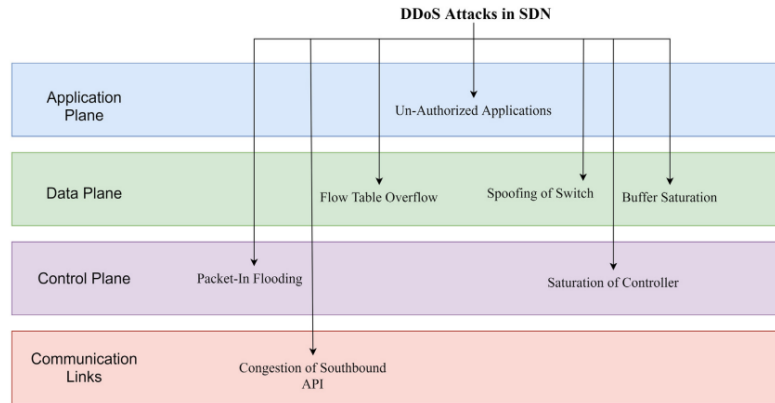


Figure II.6: Types of DDoS attacks at different layers of SDN, adopted from [9]

II.9.1 Application layer DDoS attacks

Application layer attacks maliciously use the software, aiming to exhaust resources to process any further requests. These attacks are generally harder to detect on the network level as they show no clear deviation from legitimate traffic [34]. DDoS attacks on one application will impact other applications because the separation of applications or resources in SDN is not well solved. Among the attacks of this layer:

- **Un-authorized Applications:** In the application plane, there are many applications with access to network resources for providing controller and network services. Some applications can access network resources with the example of other applications. However, the validity of applications is not authenticated and approved, and malicious applications may gain access by instances of other applications and alter network behavior and degrade network performance

II.9.2 Control layer DDoS attacks

Controllers of SDN and their communications can be subjected to different types of attacks [75]. Attacks on the control plane and communication between the controller and other network components, such as northbound API, southbound API, westbound API, or eastbound API, are examples of threats that can cause significant damage. Furthermore, the controller can be regarded as a single point of failure and scalability, raising the possibility of performance issues and control plane unavailability. Among the attacks of this layer:

- **Controller Saturation:** When multiple packet_in requests (because of fake flows generated by the attacker) come at the controller, the controller makes the queues to handle these all requests. However, suppose there are numerous fake packets. In that case, the controller will remain busy handling fake requests, and its performance will eventually degrade, which is a barrier for the SDN-based networks [76].

- **Packet_in flooding:** Whenever the switch receives the unmatched flow, it will request (using packet_in message) to the centralized controller to draw a forwarding rule for the new flow via the southbound interface. The attacker sends numerous packets to the vswitch by spoofing the IPs and forces the vswitch to send bulk packet_in messages to the controller. Resultantly, this flood overloads the controller and makes it unreachable to the legitimate users as the controller will be busy handling only fake flow requests [77].

II.9.3 Data layer DDoS attacks

Data layer DDoS attacks could potentially overload through two points: switches or by attacking the southbound API [78]. For example, an attacker may send a large amount of traffic to a node to launch a DoS attack by establishing several new and unknown flows in the infrastructure layer. Among the attacks of this layer:

- **Flow Table Overflow:** When OpenFlow switch requests for the new flow rule from the controller, the new rule sent by the controller gets store in the flow table of the switch. Every flow rule has a fixed time out value, and after that time, the old rules are evicted by the switch from the flow table [4]. TCAM is very limited in capacity to store the flow table entries because it is very costly and power-hungry [79]. The attacker takes advantage of this functionality to overwhelm the switch. The attacker sends many new fake flows to the switch, and as a result, the switch's flow table runs out of memory quickly and contains only fake rules. All valid entries are removed from the flow table, resulting in performance degradation.
- **Buffer Saturation:** When a switch sends a packet_in message to the controller, it sends a portion of the packet to the controller while storing the remainder in the buffer memory. The attacker employs this feature to attack the victim; the attacker sends multiple bogus packets to the switch, quickly depleting the buffer. In addition, when the switch internal buffering runs out, it must send the entire packet to the controllers as part of the event, yet another bottleneck for SDN. At this point, legitimate users are having difficulty processing their flow requests, and as a result, the attacker can reduce performance.
- **Spoofing Switch:** In this attack, the attacker can spoof the IP address of the switch and send control messages using the modified address. When a switch creates a connection with the controller and communicates, a second malicious switch (one with a spoofed IP and the identical hardware and name) is turned on. It starts a connection with the controller simultaneously. The controller will terminate the connection with a legitimate switch and communicates with the malicious one, gradually degrade the performance of the network [80]. This attack can cause the controller to make fake requests, and some mechanism is needed to control.

II.9.4 Communication links

between each two layers there are links which are southbound API and northbound API, among the attacks in these links:

- **Congestion of Southbound APIs:** When OpenFlow switch sends packet in request to get new rule from the controller, it sends some part of the packet, and another part is stored in the buffer. Nevertheless, if the buffer is complete, then the switch is liable to send the whole packet to the controller. At this point, by sending multiple fake flows to the switch, an attacker can easily overload the single bandwidth used in southbound APIs and create congestion in it to make it unavailable to the legitimate users.

II.10 Detecting DDoS attacks in SDN network

Distributed Denial of Service (DDoS) attacks have been a real threat in many aspects of computer networks and distributed applications. The main objective of a DDoS attack is to bring down the services of a target using multiple sources that are distributed. For example, attackers can transfer thousands of packets to a victim to overwhelm access bandwidth with illegitimate traffic, making online services unavailable. There are numerous denials of service (DoS) attack methods being used to degrade the performance or availability of targeted services on the Internet [34]. These methods can be categorized as SDN challenges at each layer: application layer, control layer, and infrastructure layer.

II.10.1 DDoS attack detection techniques

This section provides a thorough examination of current DDoS detection solutions in SDN. Depending on the type of detection metric and detection mechanism used, these solutions are classified into four categories: Information theory-based DDoS defense solutions, Machine learning-based DDoS defense solutions, Artificial Neural network-based defense solutions, and other defense solutions (see Fig. II.7). To compare these solutions, we select eight different parameters :

- year of publication
- the scope of solution (Detection or Mitigation)
- detection metric used
- parameters used in the algorithm for detection
- target plane
- controller type
- the dataset used for validation and other key features of the solution.

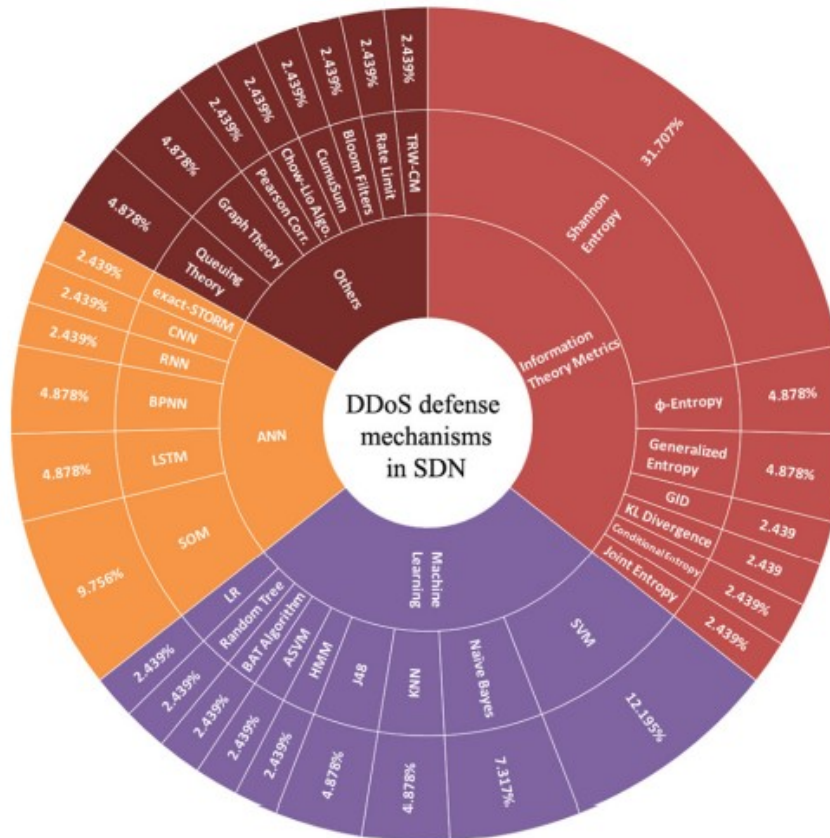


Figure II.7: Classification of DDoS attack defence approaches in SDN, adopted from [9].

Information theory-based DDoS defence solutions in SDN

Information theory-based Entropy and divergence metrics are commonly used to detect DDoS attacks. Entropy represents the randomness in the network features, whereas a divergence metric represents the similarity of two probability distributions [81]. The concept of uncertainty's measurement is coined initially by Claude Shannon in 1948 [81]. The information distance or divergence metric, which is calculated using different probability distributions of traffic flows, is used to detect network traffic anomalies. Using the entropy measure, it is possible to see how current network behavior deviates from normal network behavior, resulting in the detection of a DDoS attack.

Many researchers provided DDoS defense approaches based on the entropy metric, as shown in Table II.10.1 and discussed briefly below. Because SDN networks are programmable, they allow for the extraction and analysis of network flow statistics. Giotis et al. [82] extend this feature by reducing the controller's load when extracting the information. The proposed method collects and analyzes data using the entropy method to detect network anomalies. The collector module is in charge of collecting data regularly and sending it to the anomaly detection module. Following that, anomaly detection examines all flow entries for each time window to identify malicious flows.

Following the detection mitigation module, flow rules about malicious flows were implemented to block them. They test their method by collecting innocuous traffic on the network of the National Technical University of Athens. the Tcpreplay and Scapy programs are used to produce malicious traffic.

Machine Learning based DDoS defense solutions in SDN

In several fields, machine learning techniques are utilized to address complicated issues, these algorithms have also been used to detect DDoS attacks, and they have proven to be more effective than signature-based detection systems. These machine learning-based classifiers can be trained to detect aberrant network traffic behavior with greater accuracy. Support Vector Machine (SVM), Decision Tree (J48), Logistic regression, Advanced Support Vector Machine (SVM), Naive Bayes, Random Trees, Binary Bat algorithm, Random forest, Hidden Markov Model (HMM), and K-nearest neighbor (KNN) are some of the most commonly used machine learning classifiers, as shown and summarized in Table [II.10.1](#).

Approach reference	Scope	Classifier	Features	Dataset	Limitations
[83]	Detection	SVM	5	Synthetic + DARPA1999	Unable to indicate the attacker.
[84]	Detection Mitigation	RF	3	UCLA + Synthetic	Not accurate.
[85]	Detection	SVM	6	Synthetic	Unable to indicate the attacker.
[86]	Detection	Advanced SVM	5	Synthetic	Unable to indicate the attacker.
[87]	Detection Mitigation	J48 RF SVM K-NN	24	Synthetic	Overhead.
[88]	Detection Mitigation	SVM J48 NB	25	NSLKDD	Overhead.

Table II.2: DDoS defense solutions in SDN.

II.11 Discussion

Our comparative study of solutions for detecting DDoS attacks in the SDN networks that we chose allowed us to create table [II.10.1](#). From that, we were able to select a set of findings.

The above table [II.10.1](#) shows that the approaches have been used differently for the same purpose. In other words, while Information theory-based mechanisms collect and analyze information from switches to build a module for DDoS detection, ML-based mechanisms identify DDoS attacks by capturing the traffic and identifying appropriate features.

From the table shown previously, we could see that the studied works based on information theory approaches are efficient in estimating the randomness of a data set. Therefore, low entropy levels indicate a more concentrated to identifying probable network anomalies probability distribution. In contrast, high entropy values indicate a more distributed probability distribution. However, we argue that the full potential of entropy-based DDoS attacks detection is currently not being exploited because of its inefficient use. The machine learning approach increases the accuracy of DDoS detection and provides independence from the hardware. Moreover, it reduces workload and time. We let the algorithm perform analyzing processes on our behalf by automating tasks. Similarly, many elements affect the efficiency of machine learning. Data processing is one of them. It is capable of handling any type of data.

II.12 Conclusion

The emergence of SDN environments has made many researchers and industries look favorably on this new concept. However, it also brings some new problems to be treated, such as new kinds of DDoS attacks. For example, the DDoS attack in the SDN environment could attack the centralized controller and put down all the networks if the controller crashes. As a result, we provide a comprehensive review of the state-of-the-art DDoS attack detection strategies featured by SDN technologies. In this work, we propose the ML-Algorithms to detect DDoS attacks.

Chapter III

Proposed approach and implementation

III.1 Introduction

The previous chapter carried out a comparative study between the approaches to detect DDoS attacks in SDN networks. This chapter aims to present our approach used for our mechanism, which detects and mitigates DDoS attacks in SDN networks, and describes the tools used to implement and simulate the experimentation. The developed module for the detection and the mitigation is implemented as a Ryu controller application. The simulation of the topology is done with Mininet virtual network. For the DDoS traffic, we use the Hping3 tool.

III.2 Proposed approach

In our approach, we propose to implement machine-learning algorithms to detect and mitigate the DDoS traffic, due to its advantages. We can explain our approach with these summarized steps:

Model Building phase:

The purpose of this phase is to build our model and to ensure that we have taken the best choices and decisions while building it. Especially when we have many choices regarding dataset and machine learning algorithms. As the flow chart in figure [III.1](#) shows the first step in this phase is to collect the data since it plays a principal role in the performance of the machine learning model. In this step, we need to make research for the available data, which is related to our problem and compare the data based on the available articles and researchers who used and talked about it. After the data collecting we need to train this data using one of many ML algorithms that exist, to do this we need to know which are the most used algorithms in this case of problems and among these algorithms which one is the best for our case. According to a study in 2020, we found that The most common ML algorithms used for IDS are Decision Tree, K-Nearest Neighbor (KNN), Artificial Neural Network (ANN), Support Vector Machine (SVM), K-Mean Clustering, Fast Learning Network, and Ensemble Methods [\[89\]](#). So due to this study, we decided to use six ML algorithms named (LR, KNN, SVM, NB, DT, RF). LR and NB were selected as one of the most linear ML algorithms used and to examine and compare their performance with the other algorithms which are the most used in the case of intrusion detection systems.

The third step in this phase is to select the best algorithm among the sixth algorithms using evaluation metrics. Once we have the best model now we should use one of the feature selection techniques that we discussed in the previous chapter in order to reduce the time of prediction and avoid overfitting if it exists.

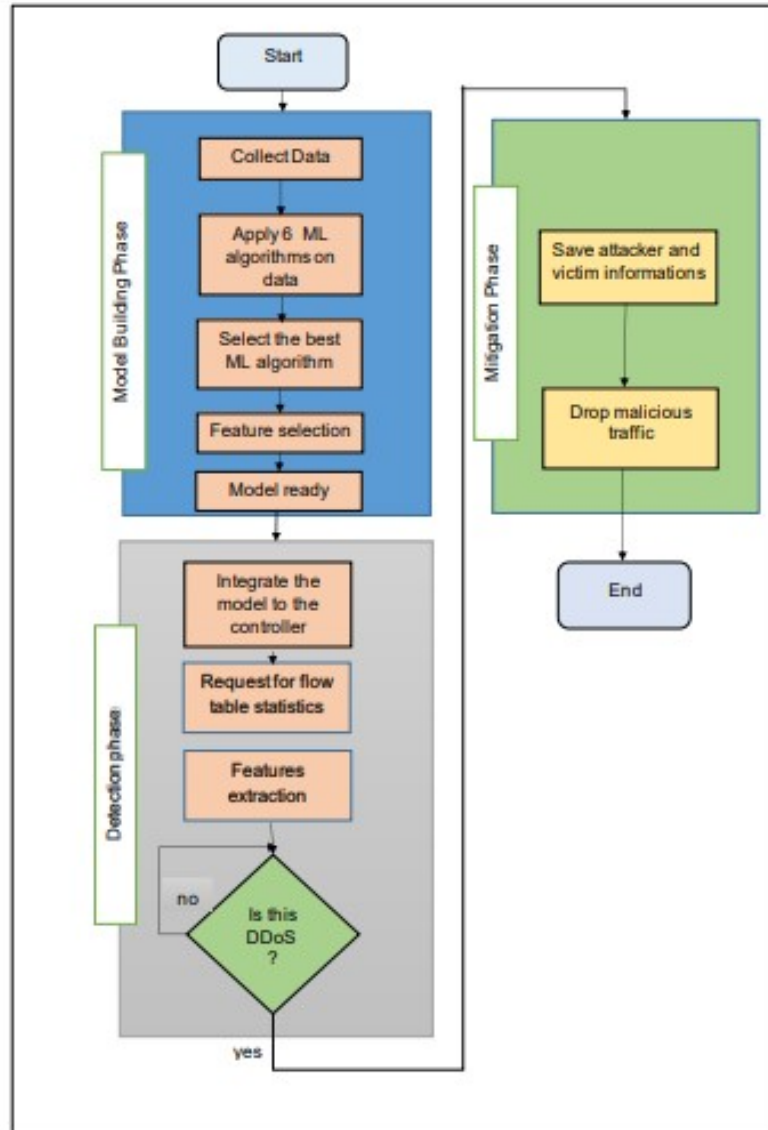


Figure III.1: The proposed approach.

Detection phase:

Once we have, our model all that we need is to integrate it into the controller so we can make the prediction and detect any anomaly on the network. Then we configure the controller to request flow table statistics every 5 seconds as used by [90] who determined that that window is the most suitable to enable detection of attacks earlier enough to protect the controller. So here the controller make features extraction to predict whether we are under attack or not.

Mitigation phase:

Once the model detects a DDoS attack it saves the information of both attacker and victim so it saves the IP address of the victim and MAC address and the datapath_ID (the switch ID) of

the attacker. Then we install a new rule to the switch we already saved its ID in order to delete all the packets coming from the attacker MAC address. In this way, we prevent any traffic coming from the attacker.

In this approach, we aim to compare between six different supervised algorithms in order to choose the best one of them. We also focus on the type of topology and how it can affect the DDoS attack to know if the topology matter and if the success percentage of this attack depend on the topology or not.

In this approach, we configure the controller to monitor the traffic and analyze it by extracting the information of each flow then use the model that we trained to make the prediction and inform as whether we are under attack or it is a legitimate traffic. Once it detect DDoS attack it adds a new flow rule to the switch that drops all the packets coming from the attacker node to mitigate and prevent this attack.

III.3 How the proposed approach works

The system works every time the controller requests the switches to send back the statistics of flow tables every five seconds. Then the switches resend these statistics to the controller. The controller uses these statistics to predict whether the flows are legitimate or it is DDoS traffic. If the traffic is predicted as DDoS, the controller indicates the source of the attack and the victim. Then it starts the mitigation phase by adding a flow entry that matches with the information of the attacker in order to drop all these packets.

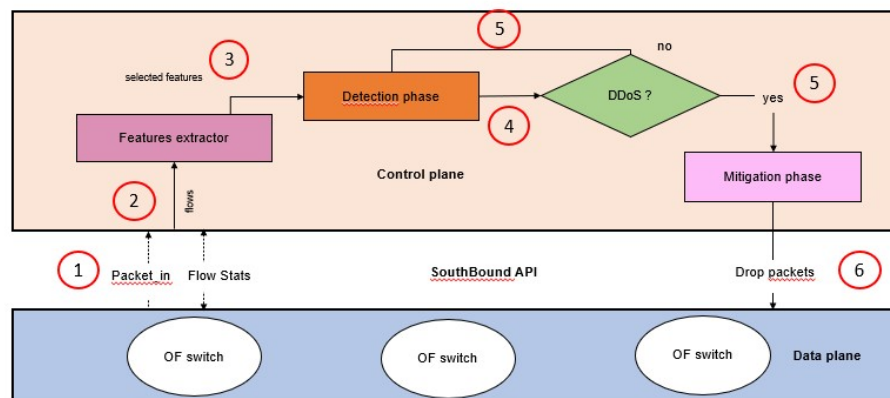


Figure III.2: Proposed approach.

The figure III.2 explain how this mechanism works. It shows how the packet_in are sent to the controller once they arrive to the switch. So here, the controller extract all the features of this packet and save it. Then before it send it to the model it makes features selection to opt the

most important features to make the prediction to reduce the time of prediction. In the detection phase, the controller use the model to make the prediction using the selected features. According to the result of prediction the controller take the decision by informing as in the two cases and in the case of attack the controller adds a new flow rule to the switch using the OpenFlow protocol to exchange this messages so the switch will drop all the packets coming from the attacker to mitigate and prevent this DDoS traffic.

III.4 Material resources

It is important to mention that this work was emulated using:

- Processor: Intel (R) Core(TM) i3-3217U CPU @ 1.80 GHz 1.80 GHz
- Installed memory (RAM): 4,00 Go
- Type of system: Operating system 64 bits, processor x64
- Windows 10 Professional

III.5 Tools selection

This section presents the tools used in this project and the reasons behind using each tool.

III.5.1 Python language

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures combined with dynamic typing and dynamic binding; make it very attractive for Rapid Application Development. In addition, python is simple, easy to learn syntax, emphasizes readability, and therefore reduces the cost of program maintenance [91].

Since we will work with machine learning and data, we found that python is widely used in this domain due to its various open-source packages.

III.5.2 Ryu Controller

Ryu is a python-based Open Flow controller with a robust API that enables developers to create their applications to manage the network. It is open-source and available under the Apache license. Moreover, it can be used to collect statistical information from switches. Thus it can be configured as a traffic monitor, a firewall, or a switch [92]. The reason behind opting Ryu from among a list of controllers to simulate our work is that:

- We are going to use the Python language to write our algorithms.
- Ryu is well documented unlike most of the other controllers.
- According to [92], [93] and [94] Ryu is very fast as compared to POX and Pyretic.

Proprieties	Ryu controller
Language	Python
OF version	1.0 , 1.1 , 1.2 , 1.3
Rest API support	yes
Platform support	Linux

Table III.1: Features of Ryu controller

III.5.3 Mininet virtual network

Mininet is an emulator which uses OpenFlow protocol, runs a collection of end-hosts, switches, routers, and links on a single Linux kernel by using virtualization. It has many tools to check the possible bandwidth, the connectivity among nodes and deepest nodes, and the speed of flows. Developers, teachers, and researchers use Mininet, and this is because of easy interaction with the network using CLI and API, customizing and sharing features, and also development features on real hardware [94].

Advantages of Mininet

- It is fast- it takes a few seconds to start a network.
- It is easy to create custom topologies ranging from a single switch to the data center topologies.
- Can run on a laptop like in this case.
- It is open source

Disadvantages of Mininet

- Supporting just one platform (Linux kernel)
- Mininet hosts share host file systems and process IDs spaces thus need extra care when running daemons requiring configuration.

The reason behind choosing Mininet among the available simulator/emulator is not just due to its advantages but also according to a comparison between these tools mentioned in [94].

III.5.4 Hping the DDoS attack tool

Hping can handle the random packet size and the fragmentations. In addition, hping performs the firewall rule testing, port scanning, and protocol-based network performance testing. Its implementation language is TCL and has a command-line interface [95].

The selection of a DDoS attack tool was tough because we found many tools used for this purpose. A comparison between different DDoS tools is provided in [96] we did not find a huge difference between them, so we opt for Hping3 randomly since it works well, and since it uses the ICMP, TCP and UDP flooding attacks which we are using in this work.

III.6 Files and functions used in the implementation

In this section, we provide an overview of the files and the functions used to implement this system before going into the depth of details. All the information are summarized in Table III.2 and Table III.3.

Access path	Files	Description
Desktop/last_files/controller	L4_switch.py	Switching applications that installs the miss entry table and switches the packet in. The match field is based on the l2, l3 and l4 in OSI model with flow expiry.
	monitor.py	This file works as a monitor and collector of normal traffic. We used it to collect normal traffic and put it in a CSV file.
	monitor_ddos.py	This file is used to monitor and collect DDoS traffic then put it in a CSV file.
	Dataset.csv	This file represents our generated dataset, it contains 23 features plus the label to make the classification.
	detector_mitigator.py	This is our application for detecting and mitigating DDoS attacks using the RF algorithm for classification and installing flow rules to the switch to drop malicious packets.
Desktop/last_files/mininet	ddos.py	This file is used to generate DDoS traffic (ICMP flood, TCP flood, UDP flood, SYN flood) with help of the Hping3 tool.
	normal.py	This file is used to generate normal traffic (ping, TCP and UDP).
	mytopo.py	This file represents our first topology scenario.
	project.py	This file represents our second topology scenario.

Table III.2: Description of used files.

Files	Functions	Description
I4_switch.py	<code>_init_()</code>	specifies which versions of the OpenFlow protocol that the application is compatible with, and initializes the internal MAC_To_Port table.
	<code>switch_features_handler()</code>	the main purpose of this code is to have it run any time a switch is added to the controller and install a table miss flow entry in the switch, which allows the switch to send packets to the controller.
	<code>add_flow()</code>	This method is defined to construct and send the final flow entry.
	<code>_packet_in_handler()</code>	called any time a switch sends a packet to the controller.
	<code>def_init_()</code>	In parallel with switching processing, initialize and define a datapaths dictionary. Open Dataset.csv file to save traffic information on it.
monitor.py	<code>_state_change_handler()</code>	Monitor the switch status, judge whether the switch is online, and then write the switch into the dictionary.
	<code>_monitor()</code>	A method to send information to the switch periodically.
	<code>_request_stats()</code>	The request method to realize the controller to request the switch, including the request flow table and port information.
	<code>_flow_stats_reply_handler()</code>	The received port information and flow table information are parsed and saved to Dataset.csv file and label the information under normal traffic.
Monitor_ddos.py	Uses the same functions of monitor.py script	The only difference between them is to save the collected information under DDoS information.
Detector_mitigator.py	<code>_init_()</code>	In parallel of switching processing, initialize and define a datapaths dictionary. Train dataset.csv file and display the information of the model.
	<code>_monitor()</code>	A method to send information to the switch periodically. Predict whether the traffic is legitimate or DDoS.
	<code>flow_training()</code>	Read the Dataset.csv file make some preprocessing on it and create the model using the RF classification model.

Table III.3: Description of the used DDoS functions.

It is important to mention that most of these functions are pre-existing in Ryu applications. The only thing that we have done is to understand the functionality of these functions, modify them and add new ones.

III.7 Setup the network

We have already mentioned two types to set up the physical connection: out-of-band and in-band. However, since we use a mininet emulator, we only limited the out-of-band connection to handle communication between controllers and switches. However, the question here is: **which topology should we use? Does the type of topology matter?**

For this reason, much research was done, but we did not find any research talking about this particularity. This encourages us to test two different topologies (tree and linear) and see the effect of a DDoS attack on each topology.

III.7.1 Tree topology

Now we need to run the controller using the switching application `l4_switch.py`. This application is required to:

- Install the Table Miss entry to the switch
- When the packet comes to Switch, it matches with Table Miss Entry, then Switch send it to the Controller (PACKET IN message)
- The controller looks at the source mac of the packet and updates it in its DB. The controller looks at the destination mac of the packet and decides on the output port.
- The controller sends the packet to switch (PACKET OUT message).
- The controller adds the flow using (FLOW Modification message), match field is based on layer 2, layer three and layer 4 with time expiry (idle and hard timeout).

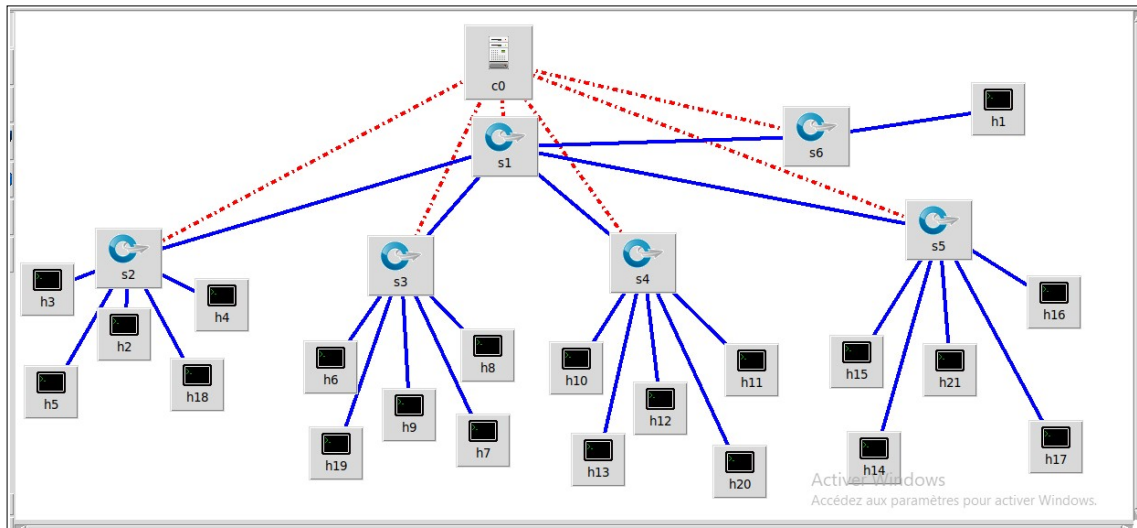


Figure III.3: Tree topology.

In this scenario, we launch the DDoS attack from host H2 (switch S2) to H15 (switch S5) the results are summarized in Table III.4, Table III.5 Table III.6 and Table III.7

switch_src	host_src	host_dst	switch_dst	Results
S2	H3	H4	S2	Unreachable destination
		H6	S3	Unreachable destination
		H14	S5	Unreachable destination
		H15	S5	Unreachable destination
		H1	S6	Unreachable destination

Table III.4: Results of pinging from H3 (switch S2) TO H4, H6, H14, H15 and H1.

switch_src	host_src	host_dst	switch_dst	Results
S3	H8	H15	S2	Unreachable destination
		H3	S3	Unreachable destination
		H12	S5	Unreachable destination
		H21	S5	Unreachable destination
		H1	S6	Unreachable destination
		H7	S3	Ping success
		H9	S3	Ping success

Table III.5: Results of pinging from h8 (switch S3) to H15, H3, H12, H21, H1, H7, H9.

switch_src	host_src	host_dst	switch_dst	Results
S4	H10	H4	S2	Unreachable destination
		H8	S3	Unreachable destination
		H12	S4	Ping success
		H15	S5	Unreachable destination
		H17	S5	Unreachable destination
		H16	S5	Unreachable destination

Table III.6: Results of pinging from H10 (switch S4) to H4, H8, H12, H15, H17, H16.

switch_src	host_src	host_dst	switch_dst	Results
S5	H16	H2	S2	Unreachable destination
		H17	S3	Ping success
		H15	S4	Ping success
		H20	S5	Unreachable destination
		H19	S5	Unreachable destination

Table III.7: Results of pinging from H16 (switch S5) to H2, H17, H15, H20, H19

From the results shown on the tables Table [III.4](#), Table [III.5](#), Table [III.6](#) and Table [III.7](#), we conclude the following:

- Reachability (connectivity) between the switches is 0% .
- All the hosts that belong to the same switch can reach each other except those belonging to the attacker switch.

This could be explained as follow:

- The attacker H2 floods the switch S2 with the malicious packets, so after some time, the flow table will not handle the received packets so that they would be lost.
- Since all the switches are connected to switch S1, the entire packet will be redirected through it and flooded with DDoS traffic. Therefore, his flow table expires, and the switches will not be able to connect.

III.7.2 Linear topology

In this scenario, we launch an attack from H2 (switch1) to H4 (switch2).all the result of the pinging test are described in tables Table [III.8](#) and Table [III.9](#).

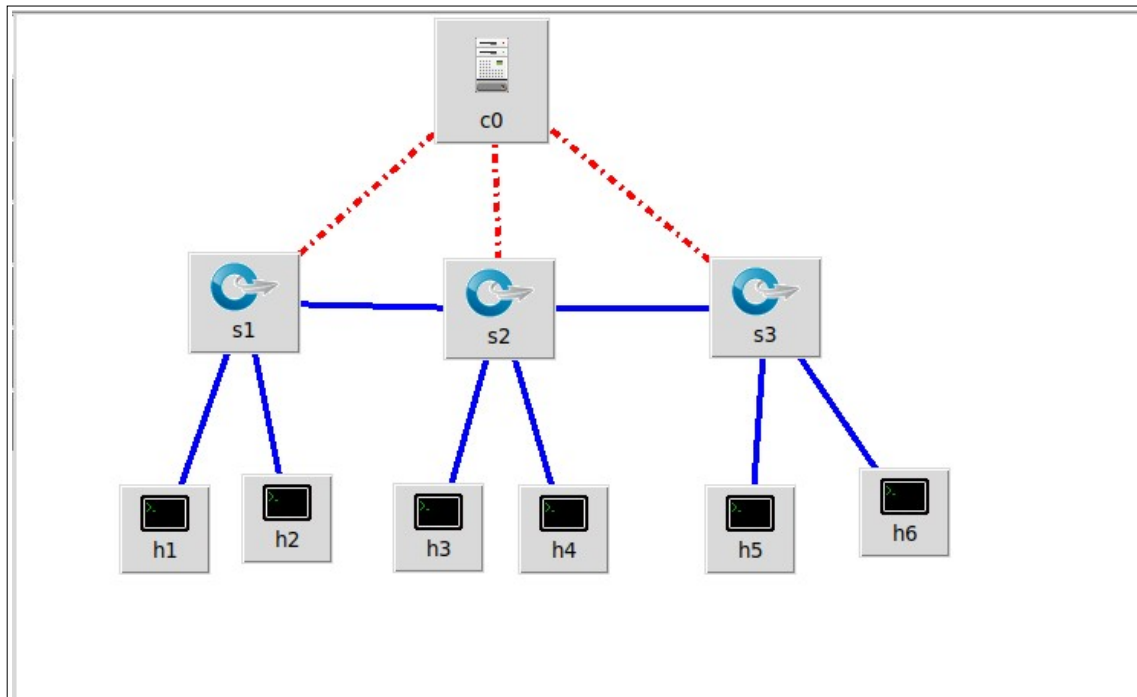


Figure III.4: Linear topology.

switch_src	host_src	host_dst	switch_dst	Results
S1	H1	H2	S4	Unreachable destination
		H3	S5	Unreachable destination
		H3	S6	Unreachable destination

Table III.8: Results of pinging from H1 (switch S3) (switch S1) to H4 and H5 and H6.

switch_src	host_src	host_dst	switch_dst	Results
S3	H6	H2	S4	Ping success
		H3	S5	Ping success
		H1	S1	Unreachable destination

Table III.9: Results of pinging from H6 (switch S3) to H4, H5, H3 and H1.

The results in Table III.8 and Table III.9 shows that:

- The connectivity between switches is 66.66
- All the hosts can connect except the hosts that belong to the switch S1.

These results can be explained as follow:

- The switch S1 in this topology becomes isolated, unlike the first topology. This is because we have multipath choices to redirect the flows.

To conclude this particularity, it is clear that the linear topology is recommended than the tree topology. This is because it can be used as the first primary step of mitigating DDoS attacks. The question answers that the topology matter and can affect the percentage of DDoS attacks' success.

The topology that we will select is the Tree topology of the first scenario. To improve that, our system for detection and mitigation of DDoS attacks is efficient regardless of the vulnerability of the topology.

III.8 Dataset and ML algorithms

This section represents the detection process building. It includes the selection of the dataset and the machine-learning algorithm for making the prediction.

III.8.1 Dataset selection

The performance of Machine Learning models vastly depends on the selection of features, quality, and quantity of training data [97].

Using a real dataset looks like a good choice at first sight since the data are collected from real scenarios in the real world. However, after some research, we found that all the datasets are retrieved from IDs, which means this data is not dedicated to DDoS, which is our study case. The second thing is that most of the datasets do not say all, are from traditional networks since we want to study an SDN network. Due to this reason, we decided to generate our dataset, which would be only for DDoS traffic and SDN networks.

In order to generate the dataset, we need to configure the Ryu controller to work as a traffic monitor, so we can gather the information and save it in a **Dataset.csv** file. The source code for generating regular traffic is in the python script **monitor.py** as it was mentioned in the previous section. (show Annexe chapter to view the source code).

To collect normal traffic we need to run `monitor.py` with our topology. We open the terminal and we run the Ryu controller as shown in figure ??.

```
amani@amani-virtual-machine:~/Desktop/last_files/controller$ ryu-manager monitor.py
loading app monitor.py
loading app ryu.controller.ofp_handler
instantiating app monitor.py of SimpleMonitor13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Figure III.5: Executing `monitor.py` script as Ryu application.

In a new terminal, we run our topology, and we launch normal traffic using **normal.py** script described in the previous section as follow:

```
amani@amani-virtual-machine:~/Desktop/last_files/controller$ ryu-manager monitor.py
loading app monitor.py
loading app ryu.controller.ofp_handler
instantiating app monitor.py of SimpleMonitor13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Figure III.6: Generating normal traffic.

We run the **normal.py** script many times with many hosts to get more rows in our **dataset.csv** file. To collect the DDoS traffic we do the same thing except that we run **monitor_ddos.py** and we launch the **ddos.py** script instead of **monitor.py** and **normal.py** respectively.

In the end, we got our dataset, which contains 3744 entries and 23 features plus the label to distinguish between normal and abnormal traffic, Our dataset uses 702.1 KB of memory. The features of the dataset are explained in Table [III.10](#).

Features	Description
timestamp	Get timestamp when Ryu received the Openflow message.
datapath_id	The switch ID.
Flow_id	Flow identifier.
ip_src	TP source address.
dL_src	MAC source address.
tp_src	TCP/UDP source port.
ip_dst	IP destination address.
dL_dst	MAC destination address.
tp_dst	TCP/UDP destination port.
ip_proto	IP protocol.
icmp_code	ICMP code.
icmp_type	ICMP type.
flow_duration_sec	Time flow was alive in seconds.
flow_duration_nsec	Time flow was alive in nano-seconds.
idle_timeout	Specifies the validity period of this entry, in seconds, if the entry is not referenced and the time specified by idle timeout elapses, that entry is deleted.
hard_timeout	Specifies the validity period of this entry, in seconds. Regardless of the reference of the entry the entry is deleted when the specified time is elapsed.
flags	Specify which operation is to be performed.
packet_count	Number of packets that were associated with the flow.
byte_count	Number of bytes that was associated with the flow.
packet_count_per_second	Number of packets per second.
packet_count_per_nsecond	Number od packets per nano-second.
byte_count_per_second	Number of bytes per second.
byte_count_per_nsecond	Number of bytes per nano-second.

Table III.10: Features description.

For the moment, we will not use feature selection. Moreover, we will keep it to the next steps.

III.8.2 Comparison between ML algorithms

As it was discussed in the previous chapter, it exists a variety of machine learning algorithm that can be used to make the classification of flows. Therefore, it was essential to compare this

subset of algorithms (LR, K-NN, SVM, NB, DT, RF) to select the highest accuracy.

For this work, we use the **ML.py** script (see Annexe I) described in the previous section.

All that we need is to run this script and select the algorithm with the best accuracy only because we know that the accuracy work well and can be satisfied when using equal number of samples belonging to each class which is our case. Let us execute the script and see the results:

```
amani@amani-virtual-machine:~/Desktop/last_files/controller$ python3 ML.py
Loading dataset ...
-----
Logistic Regression ...
-----
confusion matrix
[[ 0 116]
 [ 0 820]]
succes accuracy = 87.61 %
fail accuracy = 12.39 %
-----
LEARNING and PREDICTING Time: 0:00:00.084720
-----
K-NEAREST NEIGHBORS ...
-----
confusion matrix
[[115  1]
 [ 4 816]]
succes accuracy = 99.47 %
fail accuracy = 0.53 %
-----
LEARNING and PREDICTING Time: 0:00:00.985787
-----
SUPPORT-VECTOR MACHINE ...
-----
confusion matrix
[[ 0 116]
 [ 0 820]]
succes accuracy = 87.61 %
fail accuracy = 12.39 %
-----
LEARNING and PREDICTING Time: 0:00:00.378490
-----
```

Figure III.7: Comparison between algorithms.

```

amani@amani-virtual-machine:~/Desktop/last_files/controller$ python3 ML.py
Loading dataset ...
-----
Logistic Regression ...
-----
confusion matrix
[[ 0 116]
 [ 0 820]]
succes accuracy = 87.61 %
fail accuracy = 12.39 %
-----
LEARNING and PREDICTING Time: 0:00:00.084720
-----
K-NEAREST NEIGHBORS ...
-----
confusion matrix
[[115  1]
 [ 4 816]]
succes accuracy = 99.47 %
fail accuracy = 0.53 %
-----
LEARNING and PREDICTING Time: 0:00:00.985787
-----
SUPPORT-VECTOR MACHINE ...
-----
confusion matrix
[[ 0 116]
 [ 0 820]]
succes accuracy = 87.61 %
fail accuracy = 12.39 %
-----
LEARNING and PREDICTING Time: 0:00:00.378490
-----

```

Figure III.8: Comparison between algorithms.

The results show that:

- NB gives an accuracy of 56.62%.
- LR and SVM give the same accuracy with 87.61% of success.
- K-NN gives an accuracy of 99.47% of success.
- DT and RF give the best accuracy with 100% of success.

According to the result, we decided to go with the RF algorithm.

III.8.3 Features selection

we use feature selection in order to reduce overfitting, improves accuracy and reduce training time. We are keeping the feature selection until this part because some feature selection techniques depend on the ML algorithm, which is the case here. Since we use a tree-based model, we can obtain the best features using the feature importance technique which uses the feature importance attribute provided by the RF algorithm. This technique is straightforward, simple to understand, and fast. Feature importance technique gives you a score for each feature of your

data, the higher the score more important or relevant is the feature towards your output variable. Feature importance is an inbuilt class that comes with Tree Based Classifiers.

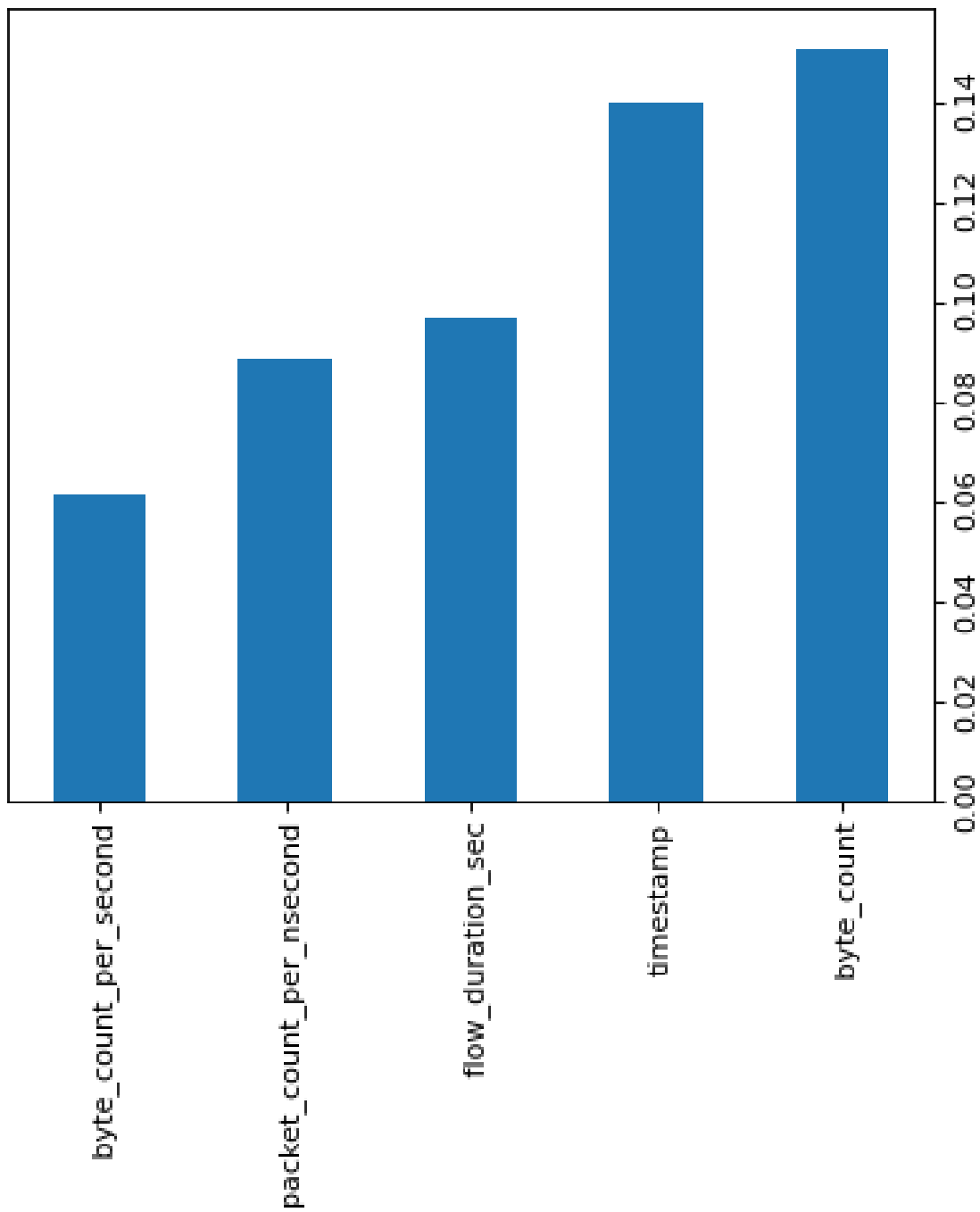


Figure III.9: Top five feature importance.

The figure III.9 shows the result of the top five most important features related to our target. The feature with the highest score is the most important feature.

The use of these five top features for training and testing reduced processing time without affecting the accuracy. The result of using full features(23 features) and the reduced features only are shown on the figure III.10 and the figure III.11 respectively.

```

X = pd.get_dummies(X, prefix_sep='_')
Y = LabelEncoder().fit_transform(Y)

X2 = StandardScaler().fit_transform(X)

X_Train, X_Test, Y_Train, Y_Test = train_test_split(X2, Y, test_size = 0.25,
                                                    random_state = 0)

start = time.process_time()
trainedforest = RandomForestClassifier(n_estimators=10).fit(X_Train,Y_Train)
print(time.process_time() - start)
predictionforest = trainedforest.predict(X_Test)
print(confusion_matrix(Y_Test,predictionforest))
print(classification_report(Y_Test,predictionforest))

```

0.07830758099999979

```

[[116  0]
 [  0 820]]

```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	116
1	1.00	1.00	1.00	820
accuracy			1.00	936
macro avg	1.00	1.00	1.00	936
weighted avg	1.00	1.00	1.00	936

Activer Win
Accédez aux pi

Figure III.10: Evaluation with full features.

```

X_Reduced = X[[X.columns[19],X.columns[22],X.columns[0] ,X.columns[20],X.columns[17]]]
X_Train2, X_Test2, Y_Train2, Y_Test2 = train_test_split(X_Reduced, Y, test_size = 0.25,
                                                        random_state = 0)

start = time.process_time()
trainedforest = RandomForestClassifier(n_estimators=10).fit(X_Train2,Y_Train2)
print(time.process_time() - start)
predictionforest = trainedforest.predict(X_Test2)
print(confusion_matrix(Y_Test2,predictionforest))
print(classification_report(Y_Test2,predictionforest))

```

0.018674422000000135

```

[[116  0]
 [  0 820]]

```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	116
1	1.00	1.00	1.00	820
accuracy			1.00	936
macro avg	1.00	1.00	1.00	936
weighted avg	1.00	1.00	1.00	936

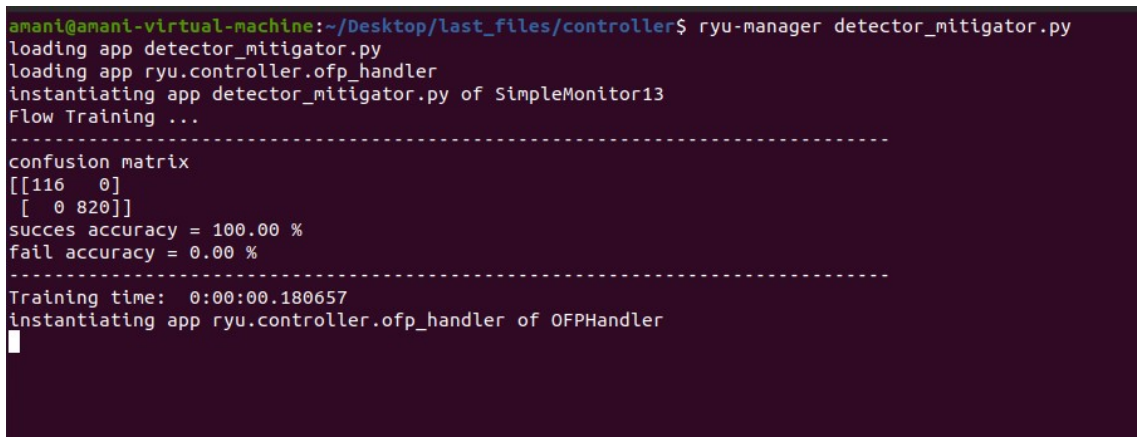
Figure III.11: Evaluation with reduced features.

III.8.4 Integration of the model with Ryu controller

In this part, we integrate our model to the Ryu controller to improve its efficiency to detect DDoS attacks in real-time. We also install flow rules to the switch to detect abnormal traffic, so the controller blocks the host attacker by dropping these packets to mitigate the attack.

This is the objective of `detector_mitigator.py` script (see Annexe I).

By running `detector_mitigator.py` as a Ryu application with `mytopo.py` topology in mininet, we get that the system is very efficient and works as it was supposed.



```
amani@amani-virtual-machine:~/Desktop/last_files/controller$ ryu-manager detector_mitigator.py
loading app detector_mitigator.py
loading app ryu.controller.ofp_handler
instantiating app detector_mitigator.py of SimpleMonitor13
Flow Training ...
-----
confusion matrix
[[116  0]
 [  0 820]]
succes accuracy = 100.00 %
fail accuracy = 0.00 %
-----
Training time: 0:00:00.180657
instantiating app ryu.controller.ofp_handler of OFPHandler
█
```

Figure III.12: Running the `detector_mitigator.py` application.

In a new terminal, we run the topology, and we launch legitimate traffic and DDoS traffic to improve the efficiency of our system.

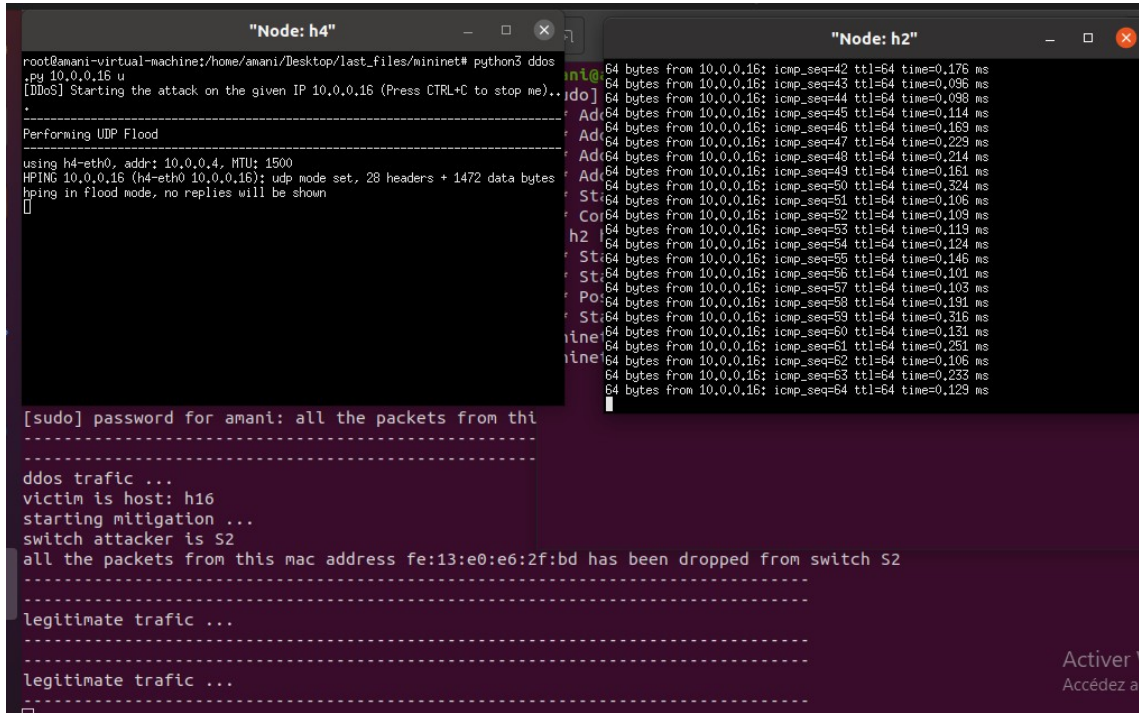


Figure III.13: Results indicating that the capability of the system to detect and mitigate DDoS attacks.

III.9 Work evaluation

Many works have detected DDoS attacks in SDN networks using the ML approach due to its advantages discussed in chapter III. In this section, we compare our work with the existing works to evaluate the selected dataset, the number of selected features, the accuracy of the model, and the proposed mechanisms to mitigate DDoS attacks.

According to [97], most of the studies have used datasets like KDD 99, NSL-KDD, DARPA, and CAIDA, etc., which excludes recent DDoS attacks. Therefore, in this comparison, we focus on the works that used one of these datasets or used synthetic datasets (generated) as in our case.

The Table III.11 represents the results summary of our work:

	Scope	Classifier	Features	Dataset	Accuracy	Detect attacker
Our approach	Detection Mitigation	RF	5 features	Synthetic	100%	yes

Table III.11: Work summary.

From Table ?? and Table III.11, it is clear that the proposed work with the use of only five features can detect the DDoS attack with an accuracy of 100%, which is perfect accuracy. In this

work, we also care about preventing the attack and we propose two mitigation mechanisms:

- Based on the topology, when we improve that, the linear topology can mitigate the DDoS attack and isolate the switch attacker from the network.
- Based on adding flow to the switch, the attacker to drop DDoS traffic coming from the attacker host (it is better to say prevention of the attack).

The prevention of the attack is done even if the attacker spoof its IP address

III.10 Conclusion

This chapter proposed an ML-based mechanism to detect DDoS attacks and demonstrated that it could detect and mitigate them accurately. Furthermore, we compared the performance of the various supervised classifier and found the Random Forest Classifier approach gives a better result.

In this chapter, we improve the capability of the proposed mechanism to detect DDoS attacks using an RF classifier with an accuracy of 100%. We also demonstrate the importance of the topology to mitigate these attacks in SDN, which is a new contribution in this domain. The simulation shows that the mitigation of DDoS attacks is efficient whatever the topology proposed.

Conclusion and Future Work

In this memory, we proposed a mechanism to detect and mitigate DDoS attacks in SDN networks. Six ML algorithms (LR, K-NN, NB, SVM, DT, and RF) were tested and evaluated using the synthetic dataset of 23 features. The results show that NB gives the worst accuracy of 56.62%, LR and SVM give the same accuracy with 87.61% of success. In the second place, K-NN gives 99.47% of success, while DT and RF give the best accuracy with 100% of success. According to the results, the RF classifier was selected using five features only. Although this is to avoid the over-fitting of the model and reduce the time of the process, the results of reducing the features show that the time was seven times less than using the full features. At the same time, the accuracy was not affected, and this is due to the quantity of the data used.

This thesis also demonstrates that the type of topology is significant in this kind of attack in SDN networks. This was provided by using a mininet virtual network to set up two topologies: tree topology and linear topology. Then we studied the effect of the DDoS attack on both topologies using the Hping3 tool. The analysis of the results indicates that the connectivity between switches in linear topology is 66.66%, unlike the tree topology, which gives 0% of connectivity between switches. Therefore, we conclude that the linear topology can mitigate the effect of DDoS attacks on SDN, unlike the tree topology, which represents a single point of failure.

While the topology cannot satisfy to mitigate the DDoS attacks, we propose using collected information from the attacker in the detection phase to add a flow rule that drops all the packets matching with the MAC address of the attacker. This mechanism proves its efficiency to prevent the attack even with a tree topology.

As part of future works of this memory, and since the mitigation part has some limitations, such as the ability of an attacker to spoof its MAC address, which means the flow rule will be unable to block the attacker. Since the penalty is stringent, the attacker host will not connect except if the administrator deletes this flow rule. We think to investigate more time to solve these problems in our system. We have thought of some solutions to solve these issues, but time was one of the challenges against achieving this, so we keep it for future ameliorations.

Bibliography

- [1] A. Maleki, M. Hossain, J.-P. Georges, E. Rondeau, and T. Divoux, “An sdn perspective to mitigate the energy consumption of core networks – geant2,” 09 2017.
- [2] P. byNutaneer, “Building blocks of sdn network,” Feb 2016.
- [3] S. H. Haji, S. R. Zeebaree, R. H. Saeed, S. Y. Ameen, H. M. Shukur, N. Omar, M. A. Sadeeq, Z. S. Ageed, I. M. Ibrahim, and H. M. Yasin, “Comparison of software defined networking with traditional networking,” *Asian Journal of Research in Computer Science*, pp. 1–18, 2021.
- [4] R. Kandoi and M. Antikainen, “Denial-of-service attacks in openflow sdn networks,” in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 1322–1326, IEEE, 2015.
- [5] B. Goswami, “Software defined network, controller comparison,” 10 2017.
- [6] M. A. M. Yusof, F. H. M. Ali, and M. Y. Darus, “Detection and defense algorithms of different types of ddos attacks,” *International Journal of Engineering and Technology*, vol. 9, no. 5, p. 410, 2017.
- [7] W. Braun and M. Menth, “Software-defined networking using openflow: Protocols, applications and architectural design choices,” *Future Internet*, vol. 6, pp. 302–336, 05 2014.
- [8] A. Thakkar and R. Lohiya, “Attack classification using feature selection techniques: a comparative study,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 1, pp. 1249–1266, 2021.
- [9] M. P. Singh and A. Bhandari, “New-flow based ddos attacks in sdn: Taxonomy, rationales, and research challenges,” *Computer Communications*, vol. 154, pp. 509–527, 2020.
- [10] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, “Software-defined internet architecture: decoupling architecture from infrastructure,” in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pp. 43–48, 2012.

- [11] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.
- [12] R. Klöti, V. Kotronis, and P. Smith, "Openflow: A security analysis," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pp. 1–6, IEEE, 2013.
- [13] K. Nagase, "Software defined network application in hospital," *InImpact: The Journal of Innovation Impact*, vol. 6, no. 1, p. 1, 2016.
- [14] K. Mlitz, "Software-defined networking market size 2027." <https://www.statista.com/statistics/468636/global-sdn-market-size/>, Jul 2021.
- [15] K. Mlitz, "Enterprise ddos protection demand worldwide 2020." <https://www.statista.com/statistics/1229430/enterprise-ddos-mitigation-service-demand/>, May 2021.
- [16] I. W. Selesnick, R. G. Baraniuk, and N. C. Kingsbury, "The dual-tree complex wavelet transform," *IEEE signal processing magazine*, vol. 22, no. 6, pp. 123–151, 2005.
- [17] M. V. Pawar and J. Anuradha, "Network security and types of attacks in network," *Procedia Computer Science*, vol. 48, pp. 503–506, 2015.
- [18] P. W. Dowd and J. T. McHenry, "Network security: it's time to take it seriously," *Computer*, vol. 31, no. 9, pp. 24–28, 1998.
- [19] B. Sotomayor and L. Childers, "Fundamental security concepts," *Globus Toolkit 4: Programming Java Services*, pp. 257–269, 2005.
- [20] D. Branstad, "Encryption protection in computer data communications," in *4th Data Communications Symposium; October 7-9, 1975; Quebec City, Quebec, Canada*, pp. 8–1, IEEE, 1975.
- [21] K. Salah, K. Elbadawi, and R. Boutaba, "Performance modeling and analysis of network firewalls," *IEEE Transactions on network and service management*, vol. 9, no. 1, pp. 12–21, 2011.
- [22] X. Liang and Y. Xiao, "Game theory for network security," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 472–486, 2012.
- [23] R. Smeliansky, "Sdn for network security," in *2014 International Science and Technology Conference (Modern Networking Technologies)(MoNeTeC)*, pp. 1–5, IEEE, 2014.
- [24] A. Voellmy, H. Kim, and N. Feamster, "Procera: a language for high-level reactive network control," in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 43–48, 2012.

- [25] T. Ubale and A. K. Jain, "Survey on ddos attack techniques and solutions in software-defined network," in *Handbook of computer networks and cyber security*, pp. 389–419, Springer, 2020.
- [26] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 3, pp. 105–110, 2008.
- [27] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE communications surveys & tutorials*, vol. 16, no. 4, pp. 1955–1980, 2014.
- [28] S. Ahmad and A. H. Mir, "Scalability, consistency, reliability and security in sdn controllers: A survey of diverse sdn controllers," *Journal of Network and Systems Management*, vol. 29, no. 1, pp. 1–59, 2021.
- [29] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain sdn controllers," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–4, IEEE, 2014.
- [30] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 1–6, 2014.
- [31] V. Thirupathi, C. Sandeep, N. Kumar, and P. Kumar, "A comprehensive review on sdn architecture, applications and major benefits of sdn," *International Journal of Advanced Science and Technology*, vol. 28, no. 20, pp. 607–614, 2019.
- [32] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013.
- [33] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an openflow architecture," in *2011 23rd International Teletraffic Congress (ITC)*, pp. 1–7, IEEE, 2011.
- [34] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "Sdn security: A survey," in *2013 IEEE SDN For Future Networks and Services (SDN4FNS)*, pp. 1–7, IEEE, 2013.
- [35] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 55–60, 2013.
- [36] M. Antikainen, T. Aura, and M. Särelä, "Spook in your network: Attacking an sdn with a compromised openflow switch," in *Nordic conference on secure IT systems*, pp. 229–244, Springer, 2014.

- [37] K. Benton, L. J. Camp, and C. Small, "Openflow vulnerability assessment," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 151–152, 2013.
- [38] O. S. Specification, "Open networking foundation," *Version ONF TS-015*, vol. 1, no. 3, pp. 1–164, 2013.
- [39] Y.-W. Chen, J.-P. Sheu, Y.-C. Kuo, and N. Van Cuong, "Design and implementation of iot ddos attacks detection system based on machine learning," in *2020 European Conference on Networks and Communications (EuCNC)*, pp. 122–127, IEEE, 2020.
- [40] M. Eslahi, R. Salleh, and N. B. Anuar, "Bots and botnets: An overview of characteristics, detection and challenges," in *2012 IEEE International Conference on Control System, Computing and Engineering*, pp. 349–354, IEEE, 2012.
- [41] C. Li, W. Jiang, and X. Zou, "Botnet: Survey and case study," in *2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*, pp. 1184–1187, IEEE, 2009.
- [42] J.-S. Lee, H. Jeong, J.-H. Park, M. Kim, and B.-N. Noh, "The activity analysis of malicious http-based botnets using degree of periodic repeatability," in *2008 International Conference on Security Technology*, pp. 83–86, IEEE, 2008.
- [43] R. Vishwakarma and A. K. Jain, "A survey of ddos attacking techniques and defence mechanisms in the iot network," *Telecommunication systems*, vol. 73, no. 1, pp. 3–25, 2020.
- [44] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.
- [45] A. Asosheh and N. Ramezani, "A comprehensive taxonomy of ddos attacks and defense mechanism applying in a smart classification," *WSEAS Transactions on Computers*, vol. 7, no. 4, pp. 281–290, 2008.
- [46] A. Bhardwaj, G. Subrahmanyam, V. Avasthi, H. Sastry, and S. Goundar, "Ddos attacks, new ddos taxonomy and mitigation solutions—a survey," in *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pp. 793–798, IEEE, 2016.
- [47] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *2019 International Carahan Conference on Security Technology (ICCST)*, pp. 1–8, IEEE, 2019.
- [48] A. Sahi, D. Lai, Y. Li, and M. Diykh, "An efficient ddos tcp flood attack detection and prevention system in a cloud environment," *IEEE Access*, vol. 5, pp. 6036–6048, 2017.

- [49] M. Xia, W. Lu, J. Yang, Y. Ma, W. Yao, and Z. Zheng, "A hybrid method based on extreme learning machine and k-nearest neighbor for cloud classification of ground-based visible cloud image," *Neurocomputing*, vol. 160, pp. 238–249, 2015.
- [50] E. Alomari, S. Manickam, B. B. Gupta, S. Karuppayah, and R. Alfaris, "Botnet-based distributed denial of service (ddos) attacks on web servers: classification and art," *arXiv preprint arXiv:1208.0403*, 2012.
- [51] R. Santos, D. Souza, W. Santo, A. Ribeiro, and E. Moreno, "Machine learning algorithms to detect ddos attacks in sdn," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 16, p. e5402, 2020.
- [52] I. El Naqa and M. J. Murphy, "What is machine learning?," in *machine learning in radiation oncology*, pp. 3–11, Springer, 2015.
- [53] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, "A survey of machine learning for big data processing," *EURASIP Journal on Advances in Signal Processing*, vol. 2016, no. 1, pp. 1–16, 2016.
- [54] F. Hussain, R. Hussain, S. A. Hassan, and E. Hossain, "Machine learning in iot security: Current solutions and future challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1686–1721, 2020.
- [55] C. Wirth, R. Akrouf, G. Neumann, J. Fürnkranz, *et al.*, "A survey of preference-based reinforcement learning methods," *Journal of Machine Learning Research*, vol. 18, no. 136, pp. 1–46, 2017.
- [56] E. Fix and J. L. Hodges, "Discriminatory analysis. nonparametric discrimination: Consistency properties," *International Statistical Review/Revue Internationale de Statistique*, vol. 57, no. 3, pp. 238–247, 1989.
- [57] Y. Akbulut, A. Sengur, Y. Guo, and F. Smarandache, "Ns-k-nn: Neutrosophic set-based k-nearest neighbors classifier," *Symmetry*, vol. 9, no. 9, p. 179, 2017.
- [58] A. E. Mohamed, "Comparative study of four supervised machine learning techniques for classification," *International Journal of Applied*, vol. 7, no. 2, 2017.
- [59] Y. Altuntaş, A. F. Kocamaz, Z. CÖMERT, R. Cengiz, and M. Esmeray, "Identification of haploid maize seeds using gray level co-occurrence matrix and machine learning techniques," in *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*, pp. 1–5, IEEE, 2018.
- [60] C. Zafer, "Fusing fine-tuned deep features for recognizing different tympanic membranes," *Biocybernetics and Biomedical Engineering*, vol. 40, no. 1, pp. 40–51, 2020.

- [61] A. Diker, Z. Cömert, E. Avci, and S. Velappan, “Intelligent system based on genetic algorithm and support vector machine for detection of myocardial infarction from ecg signals,” in *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, IEEE, 2018.
- [62] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [63] M. Usman, G. Mustafa, and M. T. Afzal, “Ranking of author assessment parameters using logistic regression,” *Scientometrics*, vol. 126, no. 1, pp. 335–353, 2021.
- [64] P. S. Saini, S. Behal, and S. Bhatia, “Detection of ddos attacks using machine learning algorithms,” in *2020 7th International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 16–21, IEEE, 2020.
- [65] I. A. A. Amra and A. Y. Maghari, “Students performance prediction using knn and naïve bayesian,” in *2017 8th International Conference on Information Technology (ICIT)*, pp. 909–913, IEEE, 2017.
- [66] A. Chapman, E. Simperl, L. Koesten, G. Konstantinidis, L.-D. Ibáñez, E. Kacprzak, and P. Groth, “Dataset search: a survey,” *The VLDB Journal*, vol. 29, no. 1, pp. 251–272, 2020.
- [67] A. Paullada, I. D. Raji, E. M. Bender, E. Denton, and A. Hanna, “Data and its (dis) contents: A survey of dataset development and use in machine learning research,” *arXiv preprint arXiv:2012.05345*, 2020.
- [68] “Real data definition.” <https://www.lawinsider.com/dictionary/real-data>.
- [69] R. Heyburn, R. R. Bond, M. Black, M. Mulvenna, J. Wallace, D. Rankin, and B. Cleland, “Machine learning using synthetic and real data: similarity of evaluation metrics for different healthcare datasets and for different algorithms,” in *Data Science and Knowledge Engineering for Sensing Decision Support: Proceedings of the 13th International FLINS Conference (FLINS 2018)*, pp. 1281–1291, World Scientific, 2018.
- [70] M. A. Hall and L. A. Smith, “Feature selection for machine learning: comparing a correlation-based filter approach to the wrapper.,” in *FLAIRS conference*, vol. 1999, pp. 235–239, 1999.
- [71] A. Thakkar and R. Lohiya, “A review of the advancement in intrusion detection datasets,” *Procedia Computer Science*, vol. 167, pp. 636–645, 2020.
- [72] V. R. Balasaraswathi, M. Sugumaran, and Y. Hamid, “Feature selection techniques for intrusion detection using non-bio-inspired and bio-inspired optimization algorithms,” *Journal of Communications and Information Networks*, vol. 2, no. 4, pp. 107–119, 2017.

- [73] 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 communications surveys & tutorials*, vol. 18, no. 1, pp. 602–622, 2015.
- [74] T. Ubale and A. K. Jain, "Taxonomy of ddos attacks in software-defined networking environment," in *International Conference on Futuristic Trends in Network and Communication Technologies*, pp. 278–291, Springer, 2018.
- [75] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "Ddos attack protection in the era of cloud computing and software-defined networking," *Computer Networks*, vol. 81, pp. 308–319, 2015.
- [76] P. Zhang, H. Wang, C. Hu, and C. Lin, "On denial of service attacks in software defined networks," *IEEE Network*, vol. 30, no. 6, pp. 28–33, 2016.
- [77] A. Akhuzada, E. Ahmed, A. Gani, M. K. Khan, M. Imran, and S. Guizani, "Securing software defined networks: taxonomy, requirements, and open issues," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 36–44, 2015.
- [78] Q. Yan and F. R. Yu, "Distributed denial of service attacks in software-defined networking with cloud computing," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 52–59, 2015.
- [79] E. S. David, D. Taylor, and J. Turner, "Packet classification using extended tcams," in *in Proceedings of IEEE International Conference on Network Protocols (ICNP)*, Citeseer, 2000.
- [80] J. M. Dover, "A denial of service attack against the open floodlight sdn controller," *Dover Networks LCC, Edgewater, MD, USA*, 2013.
- [81] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.
- [82] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments," *Computer Networks*, vol. 62, pp. 122–136, 2014.
- [83] D. Li, C. Yu, Q. Zhou, and J. Yu, "Using svm to detect ddos attack in sdn network," in *IOP Conference Series: Materials Science and Engineering*, vol. 466, p. 012003, IOP Publishing, 2018.
- [84] M. J. R. Dennis and X. Li, "Machine-learning and statistical methods for ddos attack detection and defense system in software defined networks," 2018.
- [85] J. Ye, X. Cheng, J. Zhu, L. Feng, and L. Song, "A ddos attack detection method based on svm in software defined network," *Security and Communication Networks*, vol. 2018, 2018.

- [86] M. Myint Oo, S. Kamolphiwong, T. Kamolphiwong, and S. Vasupongayya, “Advanced support vector machine-(asvm-) based detection for distributed denial of service (ddos) attack on software defined networking (sdn),” *Journal of Computer Networks and Communications*, vol. 2019, 2019.
- [87] O. Rahman, M. A. G. Quraishi, and C.-H. Lung, “Ddos attacks detection and mitigation in sdn using machine learning,” in *2019 IEEE World Congress on Services (SERVICES)*, vol. 2642, pp. 184–189, IEEE, 2019.
- [88] A. Alshamrani, A. Chowdhary, S. Pisharody, D. Lu, and D. Huang, “A defense system for defeating ddos attacks in sdn based networks,” in *Proceedings of the 15th ACM International Symposium on Mobility Management and Wireless Access*, pp. 83–92, 2017.
- [89] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, “Network intrusion detection system: A systematic study of machine learning and deep learning approaches,” *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, p. e4150, 2021.
- [90] S. M. Mousavi and M. St-Hilaire, “Early detection of ddos attacks against sdn controllers,” in *2015 International Conference on Computing, Networking and Communications (ICNC)*, pp. 77–81, IEEE, 2015.
- [91] S. Raschka, “Python machine learning equation reference.” <https://www.python.org/doc/essays/blurb/>.
- [92] J. Ali, S. Lee, and B.-h. Roh, “Performance analysis of pox and ryu with different sdn topologies,” in *Proceedings of the 2018 International Conference on Information Science and System*, pp. 244–249, 2018.
- [93] K. Kaur, S. Kaur, and V. Gupta, “Performance analysis of python based openflow controllers,” 2016.
- [94] S. Rowshanrad, S. Namvarasl, V. Abdi, M. Hajizadeh, and M. Keshtgary, “A survey on sdn, the future of networking,” *Journal of Advanced Computer Science & Technology*, vol. 3, no. 2, pp. 232–248, 2014.
- [95] N. Hoque, M. H. Bhuyan, R. C. Baishya, D. K. Bhattacharyya, and J. K. Kalita, “Network attacks: Taxonomy, tools and systems,” *Journal of Network and Computer Applications*, vol. 40, pp. 307–324, 2014.
- [96] S. Behal and K. Kumar, “Characterization and comparison of ddos attack tools and traffic generators: A review.,” *Int. J. Netw. Secur.*, vol. 19, no. 3, pp. 383–393, 2017.
- [97] N. Bindra and M. Sood, “Detecting ddos attacks using machine learning techniques and contemporary intrusion detection dataset,” *Automatic Control and Computer Sciences*, vol. 53, no. 5, pp. 419–428, 2019.

Appendix A

Deposit of the source code

Code source « monitor.py »

```
1
2 import l4_switch
3 from ryu.controller import ofp_event
4 from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
5 from ryu.controller.handler import set_ev_cls
6 from ryu.lib import hub
7 import csv
8 from datetime import datetime
9
10 class SimpleMonitor13(l4_switch.SimpleSwitch13):
11
12     def __init__(self, *args, **kwargs):
13         super(SimpleMonitor13, self).__init__(*args, **kwargs)
14         self.datapaths = {}
15         self.monitor_thread = hub.spawn(self._monitor)
16
17         file0 = open("Dataset.csv", "w")
18         file0.write('timestamp, datapath_id, flow_id, ip_src, dl_src, tp_src,
19 ip_dst, dl_dst, tp_dst, ip_proto, icmp_code, icmp_type, flow_duration_sec,
20 flow_duration_nsec, idle_timeout, hard_timeout, flags, packet_count,
21 byte_count, packet_count_per_second, packet_count_per_nsecond,
22 byte_count_per_second, byte_count_per_nsecond, label\n')
23         file0.close()
24
25     @set_ev_cls(ofp_event.EventOFPPStateChange,
26                 [MAIN_DISPATCHER, DEAD_DISPATCHER])
27     def _state_change_handler(self, ev):
28         datapath = ev.datapath
29         if ev.state == MAIN_DISPATCHER:
30             if datapath.id not in self.datapaths:
```

```

27         self.logger.debug('register datapath: %016x', datapath.id)
28         self.datapaths[datapath.id] = datapath
29     elif ev.state == DEAD_DISPATCHER:
30         if datapath.id in self.datapaths:
31             self.logger.debug('unregister datapath: %016x', datapath.id)
32             del self.datapaths[datapath.id]
33
34     def _monitor(self):
35         while True:
36             for dp in self.datapaths.values():
37                 self._request_stats(dp)
38                 hub.sleep(10)
39
40     def _request_stats(self, datapath):
41         self.logger.debug('send stats request: %016x', datapath.id)
42         parser = datapath.ofproto_parser
43
44         req = parser.OFPFlowStatsRequest(datapath)
45         datapath.send_msg(req)
46         @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
47     def _flow_stats_reply_handler(self, ev):
48         body = ev.msg.body
49
50         timestamp = datetime.now()
51         timestamp = timestamp.timestamp()
52         icmp_code = -1
53         icmp_type = -1
54         tp_src = 0
55         tp_dst = 0
56
57         file0 = open("Dataset.csv", "a+")
58
59         body = ev.msg.body
60         for stat in sorted([flow for flow in body if (flow.priority == 1)],
61 key=lambda flow:
62                             (flow.match['eth_type'], flow.match['ipv4_src'],
63 flow.match['in_port'], flow.match['eth_src'],
64                             flow.match['eth_dst'], flow.
65 match['ipv4_dst'], flow.match['ip_proto'],)):
66
67             ip_src = stat.match['ipv4_src']
68             ip_dst = stat.match['ipv4_dst']
69             ip_proto = stat.match['ip_proto']
70             dl_src= stat.match['eth_src']
71             dl_dst= stat.match['eth_dst']
72             if stat.match['ip_proto'] == 1:
73                 icmp_code = stat.match['icmpv4_code']

```

```

71         icmp_type = stat.match['icmpv4_type']
72
73     elif stat.match['ip_proto'] == 6:
74         tp_src = stat.match['tcp_src']
75         tp_dst = stat.match['tcp_dst']
76
77     elif stat.match['ip_proto'] == 17:
78         tp_src = stat.match['udp_src']
79         tp_dst = stat.match['udp_dst']
80
81     flow_id = str(ip_src) + str(tp_src) + str(ip_dst) + \
82             str(tp_dst) + str(ip_proto)
83
84     try:
85         packet_count_per_second = stat.packet_count/stat.duration_sec
86         packet_count_per_nsecond = stat.packet_count/stat.
duration_nsec
87     except:
88         packet_count_per_second = 0
89         packet_count_per_nsecond = 0
90
91     try:
92         byte_count_per_second = stat.byte_count/stat.duration_sec
93         byte_count_per_nsecond = stat.byte_count/stat.duration_nsec
94     except:
95         byte_count_per_second = 0
96         byte_count_per_nsecond = 0
97
98     file0.write("
99         .format(timestamp, ev.msg.datapath.id, flow_id,
ip_src, dl_src, tp_src, ip_dst, dl_dst, tp_dst,
100                 stat.match['ip_proto'], icmp_code, icmp_type,
101                 stat.duration_sec, stat.duration_nsec,
102                 stat.idle_timeout, stat.hard_timeout,
103                 stat.flags, stat.packet_count, stat.
byte_count,
104                 packet_count_per_second,
packet_count_per_nsecond,
105                 byte_count_per_second, byte_count_per_nsecond
, 0))
106     file0.close()

```

Source Code A.1: monitor.py.

Code source « ML.py »

```
1 from datetime import datetime
2 from matplotlib import pyplot as plt
3 import numpy as np
4 import pandas as pd
5 from sklearn.model_selection import train_test_split
6 from sklearn.linear_model import LogisticRegression
7 from sklearn.neighbors import KNeighborsClassifier
8 from sklearn.svm import SVC
9 from sklearn.naive_bayes import GaussianNB
10 from sklearn.tree import DecisionTreeClassifier
11 from sklearn.ensemble import RandomForestClassifier
12 from sklearn.metrics import confusion_matrix
13 from sklearn.metrics import accuracy_score
14 import time
15 from sklearn.preprocessing import StandardScaler
16
17 class MachineLearning():
18
19     def __init__(self):
20
21         print("Loading dataset ...")
22
23         self.counter = 0
24
25         self.flow_dataset = pd.read_csv('Dataset.csv')
26
27         self.flow_dataset.iloc[:, 2] = self.flow_dataset.iloc[:, 2].str.
replace('.', '', regex=True)
28         self.flow_dataset.iloc[:, 3] = self.flow_dataset.iloc[:, 3].str.
replace('.', '', regex=True)
29         self.flow_dataset.iloc[:, 4] = self.flow_dataset.iloc[:, 4].str.
replace(':', '', regex=True).apply(lambda x: int(x, 16))
30         self.flow_dataset.iloc[:, 6] = self.flow_dataset.iloc[:, 6].str.
replace('.', '', regex=True)
31         self.flow_dataset.iloc[:, 7] = self.flow_dataset.iloc[:, 7].str.
replace(':', '', regex=True).apply(lambda x: int(x, 16))
32
33         self.X_flow = self.flow_dataset.iloc[:, :-1].values
34         self.X_flow = self.X_flow.astype('float64')
35
36         self.y_flow = self.flow_dataset.iloc[:, -1].values
37
38         self.X_flow_train, self.X_flow_test, self.y_flow_train, self.
y_flow_test = train_test_split(
39             self.X_flow, self.y_flow, test_size=0.25, random_state=0)
```

```
40
41 def LR(self):
42
43     print(
44         "-----")
45     print("Logistic Regression ...")
46
47     self.classifier = LogisticRegression(
48         solver='liblinear', random_state=0)
49     self.Confusion_matrix()
50
51 def KNN(self):
52
53     print(
54         "-----")
55     print("K-NEAREST NEIGHBORS ...")
56
57     self.classifier = KNeighborsClassifier(
58         n_neighbors=5, metric='minkowski', p=2)
59     self.Confusion_matrix()
60
61 def SVM(self):
62
63     print(
64         "-----")
65     print("SUPPORT-VECTOR MACHINE ...")
66
67     self.classifier = SVC(kernel='rbf', random_state=0)
68     self.Confusion_matrix()
69
70 def NB(self):
71
72     print(
73         "-----")
74     print("NAIVE-BAYES ...")
75
76     self.classifier = GaussianNB()
77     self.Confusion_matrix()
78
79 def DT(self):
80
81     print(
82         "-----")
83     print("DECISION TREE ...")
84
85     self.classifier = DecisionTreeClassifier(
86         criterion='entropy', random_state=0)
```

```
87     self.Confusion_matrix()
88
89     def RF(self):
90
91         print(
92             "-----")
93         print("RANDOM FOREST ...")
94
95         self.classifier = RandomForestClassifier(
96             n_estimators=10, criterion="entropy", random_state=0)
97         self.Confusion_matrix()
98
99     def Confusion_matrix(self):
100         self.counter += 1
101
102         self.flow_model = self.classifier.fit(
103             self.X_flow_train, self.y_flow_train)
104
105         self.y_flow_pred = self.flow_model.predict(self.X_flow_test)
106
107         print(
108             "-----")
109
110         print("confusion matrix")
111         cm = confusion_matrix(self.y_flow_test, self.y_flow_pred)
112         print(cm)
113
114         acc = accuracy_score(self.y_flow_test, self.y_flow_pred)
115
116         print("succes accuracy = {0:.2f} %".format(acc*100))
117         fail = 1.0 - acc
118         print("fail accuracy = {0:.2f} %".format(fail*100))
119         print(
120             "-----")
121
122
123
124     def main():
125
126         start_script = datetime.now()
127
128         ml = MachineLearning()
129
130         start = datetime.now()
131         ml.LR()
132         end = datetime.now()
133         print("LEARNING and PREDICTING Time: ", (end-start))
```



```
134
135     start = datetime.now()
136     ml.KNN()
137     end = datetime.now()
138     print("LEARNING and PREDICTING Time: ", (end-start))
139
140     start = datetime.now()
141     ml.SVM()
142     end = datetime.now()
143     print("LEARNING and PREDICTING Time: ", (end-start))
144
145     start = datetime.now()
146     ml.NB()
147     end = datetime.now()
148     print("LEARNING and PREDICTING Time: ", (end-start))
149
150     start = datetime.now()
151     ml.DT()
152     end = datetime.now()
153     print("LEARNING and PREDICTING Time: ", (end-start))
154     start = time.process_time()
155     #start = datetime.now()
156     ml.RF()
157     print(time.process_time() - start)
158     #end = datetime.now()
159     #print("LEARNING and PREDICTING Time: ", (end-start))
160
161     end_script = datetime.now()
162     print("Script Time: ", (end_script-start_script))
163
164
165 if __name__ == "__main__":
166     main()
```

Source Code A.2: ML.py.

Code source « detector_mitigator.py »

```
1 from ryu.controller import ofp_event
2 from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
3 from ryu.controller.handler import set_ev_cls
4 from ryu.lib import hub
5
6 import l4_switch
7 from DateTime import DateTime
8 import os
```

```

9 import sys
10 import pandas as pd
11 from sklearn.model_selection import train_test_split
12 from sklearn.ensemble import RandomForestClassifier
13 from sklearn.metrics import confusion_matrix
14 from sklearn.metrics import accuracy_score
15
16 class SimpleMonitor13(l4_switch.SimpleSwitch13):
17
18     def __init__(self, *args, **kwargs):
19
20         super(SimpleMonitor13, self).__init__(*args, **kwargs)
21         self.datapaths = {}
22         self.monitor_thread = hub.spawn(self._monitor)
23
24         start = datetime.now()
25
26         self.flow_training()
27
28         end = datetime.now()
29         print("Training time: ", (end-start))
30
31     @set_ev_cls(ofp_event.EventOFPStateChange,
32               [MAIN_DISPATCHER, DEAD_DISPATCHER])
33     def _state_change_handler(self, ev):
34         datapath = ev.datapath
35         if ev.state == MAIN_DISPATCHER:
36             if datapath.id not in self.datapaths:
37                 self.logger.debug('register datapath: %016x', datapath.id)
38                 self.datapaths[datapath.id] = datapath
39             elif ev.state == DEAD_DISPATCHER:
40                 if datapath.id in self.datapaths:
41                     self.logger.debug('unregister datapath: %016x', datapath.id)
42                     del self.datapaths[datapath.id]
43
44     def _monitor(self):
45         while True:
46             for dp in self.datapaths.values():
47                 self._request_stats(dp)
48                 hub.sleep(4)
49
50             self.flow_predict()
51
52     def _request_stats(self, datapath):
53         self.logger.debug('send stats request: %016x', datapath.id)
54         parser = datapath.ofproto_parser
55

```

```

56     req = parser.OFPFlowStatsRequest(datapath)
57     datapath.send_msg(req)
58
59     @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
60     def _flow_stats_reply_handler(self, ev):
61
62         timestamp = datetime.now()
63         timestamp = timestamp.timestamp()
64
65         file0 = open("Prediction.csv","w")
66         file0.write('timestamp,datapath_id,flow_id,ip_src,dl_src,tp_src,
ip_dst, dl_dst,tp_dst,ip_proto,icmp_code,icmp_type,flow_duration_sec,
flow_duration_nsec,idle_timeout,hard_timeout,flags,packet_count,
byte_count,packet_count_per_second,packet_count_per_nsecond,
byte_count_per_second,byte_count_per_nsecond\n')
67         body = ev.msg.body
68         icmp_code = -1
69         icmp_type = -1
70         tp_src = 0
71         tp_dst = 0
72
73         for stat in sorted([flow for flow in body if (flow.priority == 1) ],
key=lambda flow:
74             (flow.match['eth_type'],flow.match['ipv4_src'],flow.match['
in_port'],flow.match['eth_src'],
75                                     flow.match['eth_dst'],flow.match
['ipv4_dst'],flow.match['ip_proto']))):
76
77             ip_src = stat.match['ipv4_src']
78             ip_dst = stat.match['ipv4_dst']
79             ip_proto = stat.match['ip_proto']
80             dl_src= stat.match['eth_src']
81             dl_dst= stat.match['eth_dst']
82
83             if stat.match['ip_proto'] == 1:
84                 icmp_code = stat.match['icmpv4_code']
85                 icmp_type = stat.match['icmpv4_type']
86
87             elif stat.match['ip_proto'] == 6:
88                 tp_src = stat.match['tcp_src']
89                 tp_dst = stat.match['tcp_dst']
90
91             elif stat.match['ip_proto'] == 17:
92                 tp_src = stat.match['udp_src']
93                 tp_dst = stat.match['udp_dst']
94
95         flow_id = str(ip_src) + str(tp_src) + str(ip_dst) + str(tp_dst) +

```

```

    str(ip_proto)
96
    try:
97
        packet_count_per_second = stat.packet_count/stat.duration_sec
98
        packet_count_per_nsecond = stat.packet_count/stat.
99
duration_nsec
    except:
100
        packet_count_per_second = 0
101
        packet_count_per_nsecond = 0
102
103
    try:
104
        byte_count_per_second = stat.byte_count/stat.duration_sec
105
        byte_count_per_nsecond = stat.byte_count/stat.duration_nsec
106
    except:
107
        byte_count_per_second = 0
108
        byte_count_per_nsecond = 0
109
110
        file0.write("
111
    {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {} \n"
112
        .format(timestamp, ev.msg.datapath.id, flow_id, ip_src,
113
            dl_src, tp_src, ip_dst, dl_dst, tp_dst,
114
            stat.match['ip_proto'], icmp_code, icmp_type,
115
            stat.duration_sec, stat.duration_nsec,
116
            stat.idle_timeout, stat.hard_timeout,
117
            stat.flags, stat.packet_count, stat.byte_count,
118
            packet_count_per_second, packet_count_per_nsecond,
119
            byte_count_per_second, byte_count_per_nsecond))
120
    file0.close()
121
def flow_training(self):
122
123
    self.logger.info("Flow Training ...")
124
125
    flow_dataset = pd.read_csv('Dataset.csv')
126
127
128
    X_flow = flow_dataset.iloc[:, [19,22,0,20,17]].values
129
    X_flow = X_flow.astype('float64')
130
131
    y_flow = flow_dataset.iloc[:, -1].values
132
133
    X_flow_train, X_flow_test, y_flow_train, y_flow_test =
134
train_test_split(X_flow, y_flow, test_size=0.25, random_state=0)
135
136
    classifier = RandomForestClassifier(n_estimators=10, criterion="
entropy", random_state=0)

```

```
137     self.flow_model = classifier.fit(X_flow_train, y_flow_train)
138
139     y_flow_pred = self.flow_model.predict(X_flow_test)
140
141     self.logger.info("-----")
142
143     self.logger.info("confusion matrix")
144     cm = confusion_matrix(y_flow_test, y_flow_pred)
145     self.logger.info(cm)
146
147     acc = accuracy_score(y_flow_test, y_flow_pred)
148
149     self.logger.info("succes accuracy = {0:.2f} %".format(acc*100))
150     fail = 1.0 - acc
151     self.logger.info("fail accuracy = {0:.2f} %".format(fail*100))
152     self.logger.info("-----")
153
154     def flow_predict(self):
155         try:
156
157             predict_flow_dataset = pd.read_csv('Prediction.csv')
158             predict_flow_dataset1 = pd.read_csv('Prediction.csv')
159
160             predict_flow_dataset1.iloc[:, 6] = predict_flow_dataset1.iloc[:,
values
161             6].str.replace('.', '', regex=True)
162
163             X_predict_flow= predict_flow_dataset.iloc[:, [19,22,0,20,17]].
164             X_predict_flow = X_predict_flow.astype('float64')
165
166             y_flow_pred = self.flow_model.predict(X_predict_flow)
167
168             legitimate_traffic = 0
169             ddos_traffic = 0
170
171             for i in y_flow_pred:
172                 if i == 0:
173                     legitimate_traffic = legitimate_traffic + 1
174                 else:
175                     ddos_traffic = ddos_traffic + 1
176                     victim = int(predict_flow_dataset1.iloc[i, 6])%20
177                     mac = str(predict_flow_dataset1.iloc[i, 4])
178                     dpid= str(predict_flow_dataset1.iloc[i, 1])
179
180
181             self.logger.info("-----")
```

```
182         if (legitimate_traffic/len(y_flow_pred)*100) > 80:
183             self.logger.info("legitimate traffic ...")
184         else:
185             self.logger.info("ddos traffic ...")
186             self.logger.info("victim is host: h{}".format(victim))
187             self.logger.info("starting mitigation ...")
188
189             print("switch attacker is S"+dpid)
190             cmd='ovs-ofctl add-flow s'+dpid+' dl_src='+mac+',actions=drop
191             ,
192             passw='user'
193             os.system('echo %s|sudo -S %s' % (passw,cmd))
194
195             self.logger.info("all the packets from this mac address "+mac
196             +" has been dropped from switch S" +dpid)
197
198             self.logger.info("-----")
199             file0 = open("Prediction.csv","w")
200
201             file0.write('timestamp,datapath_id,flow_id,ip_src,dl_src,tp_src,
202             ip_dst,dl_dst, tp_dst,ip_proto,icmp_code,icmp_type,flow_duration_sec,
203             flow_duration_nsec,idle_timeout,hard_timeout,flags,packet_count,
204             byte_count,packet_count_per_second,packet_count_per_nsec,
205             byte_count_per_second,byte_count_per_nsec\n')
```

Source Code A.3: detector_mitigator.py.