

Scalable and Reliable Framework to Detect and Mitigate DDoS Attack in OpenFlow-based SDN Network

Dissertation

zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades

"Doctor rerum naturalium"

der Georg-August-Universität Göttingen

im Promotionsprogramm Computer Science (PCS)

der Georg-August University School of Science (GAUSS)

vorgelegt von

Amirreza Fazely Hamedani

aus Teheran – Iran

Göttingen, 2023

Betreuungsausschuss

Prof. Dr. Ramin Yahyapour
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG),
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Dieter Hogrefe
Institut für Informatik, Georg-August-Universität Göttingen

Mitglieder der Prüfungskommission

Referent: Prof. Dr. Ramin Yahyapour
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG),
Institut für Informatik, Georg-August-Universität Göttingen

Korreferent: Prof. Dr. Dieter Hogrefe
Institut für Informatik, Georg-August-Universität Göttingen

Weitere Mitglieder der Prüfungskommission

Prof. Dr. Florentin Andreas Wörgötter
Third Institute of Physics, Georg-August-Universität Göttingen

Prof. Dr. Bernhard Schmitzer
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Jens Grabowski
Institut für Informatik, Georg-August-Universität Göttingen

Hon.-Prof. Dr. Philipp Wieder
Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG)

Tag der mündlichen Prüfung: 4 Juli, 2023

Acknowledgments

The completion of this thesis would not have been possible without the support and encouragement of several special people. Hence, I would like to take this opportunity to indicate my gratitude to those who have supported me in various ways.

First and foremost, I would like to express my heartfelt thanks to my supervisor Prof. Dr. Ramin Yahyapour for giving me this opportunity, for his enthusiasm, continuous support on many occasions, patient guidance, and unceasing ideas. A more supportive and considerate supervisor than I could not have asked for. I would also like to express my gratitude to my second supervisor, Prof. Dr. Dieter Hogrefe, who always delivered valuable advice for my research, despite his busy schedule being the director of the Institute of Computer Science.

Moreover, I would like to thank my amazing current and former colleagues and friends from the Gesellschaft für Wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG) for the incredible working environment and the enormous support who made this research journey enjoyable, despite the hard times which I passed through. In particular, I would like to thank Prof. Dr. Phillip Wieder for his encouragement and moral support, and Dr. Muzzamil Aziz for his support on scientific and non-scientific matters which was a significant aid in completing this work. I would like to express my gratitude to all of the eScience group members who are too numerous to mention here.

Finally, my deep and sincere gratitude to my wife Hadis for her continuous and unparalleled love, help, and support. Throughout the ups and downs of this challenging journey, she provided me with a constant source of inspiration and motivation. I am also grateful to my brother Amirhossein for always being there for me as a friend. I would like to express my heartfelt gratitude to my dear friend Azad Ahmadi, who provided unwavering support and encouragement throughout my journey in completing this thesis.

And last but not least, I am forever indebted to my late parents for giving me the opportunities and experiences that have made me who I am.

Abstract

Software-defined networking in recent years has come into the sight of so many network designers as a successor to traditional networking. Unlike traditional networks where control and data planes engage together within a single device in the network infrastructure such as switches and routers, the two planes are kept separated in software-defined networks (SDNs). All critical decisions about packet routing are made by the network controller, and the data-level devices forward the packets based on these decisions.

This type of network is vulnerable to DDoS attacks, degrading the overall functioning and performance of the network by continuously injecting fake flows into it. This increases the substantial burden on the controller side, and the result ultimately leads to the inaccessibility of the controller and the lack of network service to legitimate users. Thus, the protection of this novel network architecture against denial-of-service attacks is essential.

Today, the world is on the verge of using computer network services and information systems at their peak. Thus, the security concerns attached to these services/systems shall be taken seriously and dealt with. In the world of cybersecurity, attacks, and new threats emerge every day. It is essential to have tools capable of managing and analyzing all this new information to detect possible attacks in real-time. These tools should provide a comprehensive solution to automatically detect, predict and prevent abnormalities in the network.

Big data, though, encompasses a wide range of studies; it mainly refers to the massive amounts of structured and unstructured data that organizations deal with on a regular basis. It essentially refers to the amount of data coming from the applications being so huge that it is not possible to be captured, organized, processed, and managed in a reasonable time by the current mainstream software tools that help enterprises make business decisions. On the other hand, it regards not only the volume of the data; but also, how data-driven information can be employed to develop decision-making processes, security, and the overall efficiency of a business.

This Ph.D. research introduces an intelligent big data framework as a solution to prevent the performance degradation on SDN network during DDoS attacks. By leveraging the programmability and centralized controller of the SDN and using distributed data processing techniques based on the big data analytic tools and machine learning algorithm, we aim to enhance the SDN network security and resilience against these sophisticated attacks.

Table of Contents

Acknowledgments	1
Abstract	2
List of Acronyms	7
List of Figures.....	9
Chapter 1	13
1 Introduction	13
1.1 Challenges in SDN network security	14
1.2 Motivation.....	16
1.3 Objectives.....	19
1.3.1 Research Questions.....	19
1.3.2 Specific objectives	20
1.4 Goals and Contributions	21
1.4.1 Thesis Impact.....	22
1.5 Thesis Outline.....	24
Chapter 2	26
2 Literature Review	26
2.1 General Methods	26
2.2 Mathematical Methods	27
2.3 Machine Learning Methods	27
2.4 Big Data Methods.....	28
2.5 Hybrid Methods (Big Data and Machine Learning).....	29
2.6 Research Gap	31
2.7 Chapter Summary	32
Chapter 3	33
3 Technology Background	33
3.1 Overview	34
3.2 SDN History	34
3.3 The Concept of SDN	36
3.3.1 General definitions of SDN.....	36
3.3.2 The major characteristic of the SDN Network.....	37
3.4 SDN Architecture	38
3.4.1 The Application Layer.....	38
3.4.2 The Control Layer	39
3.4.3 The Infrastructure/Data-plane Layer	40
3.4.4 SDN Programming Interfaces	41

3.5	OpenFlow protocol	43
3.5.1	OpenFlow Architecture	46
3.5.2	OpenFlow Workflow	47
3.5.3	Flow Load Modes	50
3.6	SDN Security.....	51
3.6.1	SDN Challenges and Issues.....	52
3.6.2	SDN Vulnerabilities.....	53
3.6.3	SDN Attack Vectors	55
3.6.4	Improve Security in SDN.....	58
3.7	Denial of Service Attacks.....	59
3.7.1	Different Types of DDoS Attacks.....	62
3.7.2	DDoS Attacks on SDN Networks.....	65
3.8	Chapter Summary	67
Chapter 4		68
4	Evaluating the SDN Controllers.....	68
4.1	Features for Selecting the SDN Controllers	68
4.2	SDN Controllers.....	70
4.2.1	NOX/POX:	71
4.2.2	Ryu Controller	72
4.2.3	Floodlight Controller	73
4.2.4	OpenDaylight (ODL).....	73
4.2.5	ONOS Controller	75
4.2.6	Conclusion	75
4.3	Open Network Operating System (ONOS).....	76
4.3.1	ONOS Use Cases	77
4.3.2	Clustering capability of the ONOS.....	78
4.3.3	Security concerns in the ONOS controller.....	78
4.3.4	ONOS Overview.....	79
4.3.5	ONOS Architecture	81
4.4	Chapter Summary	86
Chapter 5		87
5	Proposed Framework (BFDD-S)	87
5.1	BFDD-S Methodology.....	87
5.1.1	Motivation behind BFDD-S.....	89
5.2	Framework Architecture	90
5.3	Framework Workflow	93
5.4	API Development for SDN controller	95
5.5	Machine Learning Algorithms Evaluation	99

5.5.1	Algorithm Methods	99
5.5.2	Process of Designing a Model in this Research	101
5.5.3	Evaluate the Accuracy of a Model.....	102
5.5.4	Classification Algorithms	105
5.5.5	Intrusion Detection Using Techniques	109
5.5.6	Data Set Selection	109
5.6	Selection of Proper Machine Learning Algorithm.....	113
5.6.1	Data Preprocessing.....	114
5.6.2	Comparison of the Classification Algorithms	117
5.7	Tools Used for Proposed Framework (BFDD-S)	120
5.7.1	General Tools	121
5.7.2	Big Data Analytics Tools	125
5.8	Implementing the Big Data Pipeline	130
5.8.1	Pre-Requisite Configuration	130
5.8.2	Implementation of Big Data Infrastructure.....	130
5.8.3	Implementing Data Pipeline.....	139
5.8.4	Implementing Machine Learning in Apache Spark.....	141
5.9	Chapter Summary	145
Chapter 6		147
6	Experimental Setup and Performance Evaluation.....	147
6.1	Setup Experimental Testbed	147
6.2	Deploying Network Topology	148
6.3	BFDD-S Framework Operation.....	151
6.4	Performance Evaluation of the BFDD-S Framework.....	154
6.4.1	Performance Evaluationof of the BFDD-S Framework with Centralized Detection Method	156
6.4.2	Performance Evaluation of the BFDD-S Framework with Traditional Methods	160
6.5	Chapter Summary	161
Chapter 7		163
7	Projects Contribution and Research Use Cases	163
7.1	Nephele Project.	163
7.1.1	Project goals.....	164
7.1.2	Our Contribution	165
7.2	SENDATE Project.	165
7.2.1	Project Goals	165
7.2.2	Our Contribution	168
7.3	AI-NET-PROTECT Project.....	168
7.3.1	Project goals.....	168

7.3.2	Work Package Three: AI-based Network control & service automation	169
7.3.3	Our Contribution	169
Chapter 8		171
8	Conclusion and Future Work	171
8.1	Introduction	171
8.2	Conclusions about the Research Questions.....	171
8.3	Research Summary	174
8.4	Future Work.....	176
Bibliography		177
Appendix A.....		190
A. The Source Code of Machine Learning Evaluation.....		190

List of Acronyms

SDN	Software-defined Networking
DoS	Denial of Service
DDoS	Distributed Denial of Service
U2R	Users-to-root attack
R2L	Remote-to-local attack
PROBE	Probing attack
IoT	Internet of Things
DNS	Domain Name Service
CPU	Central Processing Unit
API	Application Programming Interface
ML	Machine Learning
LR	Logistic Regression
KNN	k-Nearest Neighbor
NB	Naïve Bayes
DT	Decision Tree
RF	Random Forest
NETCONF	Network Configuration Protocol
IPFIX	Internet Protocol Flow Information Export
HTTP	Hypertext Transfer Protocol
ANN	Artificial Neural Network
OSI	Open System Interconnection
TCP/IP	Transmission Control Protocol/Internet Protocol
NFV	Network Function Virtualization
REST	Representational State Transfer
NOS	Network Operating System
ODL	OpenDaylight
ONOS	Open Network Operating System
OVS	Open vSwitch
OVSDB	Open vSwitch Database
CLI	Command-Line Interface
GUI	Graphical User Interface
CORD	Central Office Re-architected as a Data Center
vCPE	Virtual Customer Premises Equipment
vOLT	Virtual Optical Line Termination
NFaaS	Network Function-as-a-Service
VPLS	Virtual Private LAN Service
BGP	Border Gateway Protocol
FML	Flow-based Management Language
SNMP	Simple Network Management Protocol
LISP	Location Identifier Separation Protocol
ForCES	Forwarding and Control Element Separation
ONF	Open Network Foundation
QoS	Quality of Service
PCEP	Path Computation Element Communication Protocol

DCI	Data Center Interconnect
NVGRE	Network Virtualization using Generic Routing Encapsulation
STT	Stateless Transport Tunneling
VxLAN	Virtual Extensible LAN
MTM	Man in the Middle Attack
SSL	Secure Sockets Layer
TLS	Transport Layer Security
SOC	Security Operation Center
ICMP	Internet Control Message Protocol
UDP	User Datagram Protocol
U2R	User to Root Attack
R2L	Remote to Local Attack
RDD	Resilient Distributed Dataset
JSON	JavaScript Object Notation
OPNFV	Open Platform for NFV Project
ONAP	Open Network Automation Platform
VLAN	Virtual local area networks
VRF	Virtual routing and forwarding

List of Figures

Figure 1. 1: The rise in sophisticated DDoS attacks in recent years.	14
Figure 1. 2: Attack on SDN Layers.	17
Figure 3. 1: The Architect of the SDN Network.....	38
Figure 3. 2: Communication between Controllers via East / West APIs	42
Figure 3. 3: OpenFlow Versions Timeline	44
Figure 3. 4: OpenFlow Network Architecture	45
Figure 3. 5: SDN / OpenFlow device.	47
Figure 3. 6: OpenFlow Switch Operation workflow	48
Figure 3. 7: Exchanges the OpenFlow messages between the controller and the switch	49
Figure 3. 8: Sample flow sent to the controller via OpenFlow.	50
Figure 3. 9: SDN Security Vectors	56
Figure 3. 10: DDoS attack.....	59
Figure 3. 11: The incidence of various DDoS attacks from January 2020 to March 2021	61
Figure 3. 12: Three-step negotiation in TCP	63
Figure 3. 13: TCP Flood attack.....	64
Figure 3. 14: DDoS Attack on SDN Network Using the Botnet	65
Figure 4. 1: NOX controller architecture.....	71
Figure 4. 2: Ryu controller architecture	72
Figure 4. 3: Floodlight controller architecture	73
Figure 4. 4: ODL Controller Architecture	74
Figure 4. 5: Overview of the ONOS Architecture	75
Figure 4. 6: ONOS CLI interface.....	80
Figure 4. 7: ONOS GUI.....	80
Figure 4. 8: Overview of the OSGi framework	81
Figure 4. 9: ONOS architecture	82
Figure 4. 10: Relationship between ONOS subsystem components	83
Figure 4. 11: ONOS Subsystem	85
Figure 5. 1: Overall Architecture of the Proposed Framework.....	91
Figure 5. 2: Information Gathering and Mitigation Module.....	91
Figure 5. 3: Intrusion Detection Module.....	92
Figure 5. 4: Machine Learning Module.	93
Figure 5. 5: Five Phases of the BFDD-S Framework’s packet processing.....	95
Figure 5. 6: Overview of the Apache Karaf	96
Figure 5. 7: Maven Objectives	97
Figure 5. 8: A typical development process for an ONOS application	98
Figure 5. 9: Various Types of Machine Learning Algorithms	101
Figure 5. 10: Process of Design Machine Learning Model.....	102
Figure 5. 11: Overall View of the Confusion Matrix and Assessment Methods	103
Figure 5. 12: Three main assessment Metrics	104
Figure 5. 13: The machine learning workflow used in this research.	113
Figure 5. 14: Importing the essential Python Libraries.....	114
Figure 5. 15: Importing the Data set and Creating the Pandas Data Frame.....	115

Figure 5. 16: Data pre-processing workflow	115
Figure 5. 17: Convert Qualitative Data	116
Figure 5. 18: Applying One-Hot Encoding	116
Figure 5. 19: Splitting Dataset	117
Figure 5. 20: Data Scaling	117
Figure 5. 21: Training Machine Learning Algorithms	118
Figure 5. 22: Comparison of different ML Algorithms using the NSL-KDD dataset	119
Figure 5. 23: Comparison of different ML Algorithms using the UNSW-NB15 dataset	120
Figure 5. 24: Comparison of NSL-KDD and UNSW-NB15 datasets results for different ML Algorithms	120
Figure 5. 25: Sample Mininet CLI command	122
Figure 5. 26: Open vSwitch Architecture	123
Figure 5. 27: The Ecosystem of the Apache Spark	126
Figure 5. 28: Two main packages of Spark's MLlib library	127
Figure 5. 29: Integration of Apache Streaming with any other Spark components	128
Figure 5. 30: Kafka Architecture	129
Figure 5. 31: Running Spark in Standalone Server Mode	131
Figure 5. 32: Spark Web GUI	131
Figure 5. 33: Launching Spark Worker	132
Figure 5. 34: Spark Standalone Mode Information	132
Figure 5. 35: Spark Shell	133
Figure 5. 36: Running Zookeeper	134
Figure 5. 37: Running Kafka Server(Broker)	135
Figure 5. 38: Create a Topic in Kafka	136
Figure 5. 39: Kafka Topic Information	136
Figure 5. 40: Define Kafka Consumer	136
Figure 5. 41: Running Elasticsearch	137
Figure 5. 42: Elasticsearch Information	137
Figure 5. 43: Create Index in Elasticsearch for Storing Data	138
Figure 5. 44: Information of the Created Index in Elasticsearch	139
Figure 5. 45: Import Spark Essential Libraries	139
Figure 5. 46: Create Spark Context	140
Figure 5. 47: Process of the live input data in the Spark streaming	140
Figure 5. 48: Generates an input stream that fetches messages from the Kafka broker.	141
Figure 5. 49: Creating RDD in Spark	141
Figure 5. 50: Transforming Numerical Features into Single Vector	142
Figure 5. 51: Scaling Features in Spark	142
Figure 5. 52: Import Random Forest Classifier in Spark	142
Figure 5. 53: Define Random Forest Model in Spark	143
Figure 5. 54: REST API Architecture	144
Figure 5. 55: JSON File for Creating REST API Request	145
Figure 5. 56: Sending Request to the Controller via REST API POST Method	145
Figure 6. 1: Experimental Testbed Architecture	148
Figure 6. 2: Activate the OpenFlow and Forwarding apps in ONOS	150
Figure 6. 3: Network Setup from Controller Point of View	151
Figure 6. 4: Ping command before launching flooding attack	151
Figure 6. 5: Legitimate Traffic Detection Message	152

Figure 6. 6: Anomaly Detection Message by the Framework.....	153
Figure 6. 7: Ping command After Blocking the Attacker.....	153
Figure 6. 8: Average Controller CPU Consumption.....	154
Figure 6. 9: Average Controller Memory Consumption.....	154
Figure 6. 10: Average Controller Response time to Legitimate Traffic.....	155
Figure 6. 11: Comparison between BFDD-S and centralized method in terms of average response time to the legitimate traffic.	157
Figure 6. 12: Comparison between BFDD-S and centralized method in terms time required to mitigate DDoS attack by the controller.....	158
Figure 6. 13: Comparison between BFDD-S and centralized method in terms of processing load on the controller during the DDoS attack.....	158
Figure 6. 14: Comparison between BFDD-S and centralized method in terms of memory consumption during the DDoS attack.	159
Figure 6. 15: The average processing load in the traditional and proposed method.	160
Figure 6. 16: The average response time to the legitimate flows in the traditional defense and proposed methods.	161
Figure 7. 1: Agent Position in SDN Architecture	165
Figure 7. 2: A Simplified DCI Architecture for DCs with Automation. S: spine, L: leaf.	167
Figure 7. 3: Intermediate Orchestrator or Orchestrator Cascading.	167
Figure 7. 4: Overview of our Contribution to WP3.	169
Figure 7. 5: Overview of our Contribution to WP4	170

List of Tables

- Table 3. 1:** Functionality of the different OpenFlow specifications..... 46
- Table 3. 2:** Main Threats in the SDN Layers 55

- Table 4. 1:** SDN controllers and their characteristics 76

- Table 5. 1:** Pseudo code for the Logistic Regression algorithm 105
- Table 5. 2:** Pseudocode for the KNN Algorithm 106
- Table 5. 3:** Pseudo code for the Naïve Bayes Algorithm 107
- Table 5. 4:** Pseudo code for the Decision Tree algorithm 108
- Table 5. 5:** Pseudo code for the random forest algorithm 109
- Table 5. 6:** List of features of the NSL-KDD dataset..... 111
- Table 5. 7:** Divisions of famous and new attacks in the KDD-Test set..... 112
- Table 5. 8:** Number of records in training and testing subsets for each class..... 113
- Table 5. 9:** Accuracy Comparison Table (Using NSL-KDD data set) 118
- Table 5. 10:** Comparison Table (Using UNSW-NB15 data set)..... 119

- Table 6. 1:** Hardware and Software used in the Testbed. 148

- Table 7. 1:** Projects Contributions..... 163

Chapter 1

1 Introduction

The Computers on the World Wide Web are connected using a standard protocol called the Internet. Today, people rely on the Internet for education, business, socialization, and entertainment, among many vital aspects of human life, as well as sharing information, and e-commerce. We can claim that the Internet is the biggest revolution in the world of computing and communication. Web threats, on the other hand, cause a wide range of jeopardies including financial abduction, identity theft, loss of information or confidentiality, network resources theft, damage to the brand and personal reputation, and consumer trust decrease in e-commerce, online banking, and all different network services.

The traditional network structure initially has limitations such as complexity, inconsistent policies, lack of scalability, dependency on manufacturers, and a lack of symmetry between market demands and network capabilities. New concepts such as Cloud, virtualization, and changes in data consumption patterns, reveal weaknesses and limitations in traditional networks. With the advent of software-defined networks, new hopes have emerged for finding solutions for structural problems and resolving restrictions in traditional networks.

Software-defined networking is a dynamic, manageable, cost-effective, and adaptable architecture that seeks to be suitable for today's dynamic and high-bandwidth applications that can fundamentally overcome the restrictions of traditional networks [1]. Due to its centralized control, a software-defined network facilitates management, reduces hardware complexity, and gives the network environment more flexibility and mobility [2].

Denial of service attacks is one of the hazards that can create a serious and unexpected challenge for accessing essential information by eligible users. By performing this kind of attack, the attackers attempt to disrupt the services by occupying a significant amount of available resources [3]. Although the technical complexity of these attacks is increasing daily, the attackers do not need high technical knowledge and skills about the victim's system and attack launch techniques to perform such attacks [4].

Hackers have enhanced their DDoS attack methods to extraordinary levels over the past decade. The emergence of the internet of things (IoT), pervasive connectivity, and now 5G networks are all components that have led to DDoS attacks' rapid expansion and complexity [5]. These kinds of attacks are growing daily. The DDoS attack has evolved into a substantial danger to organizations and Internet infrastructures over the last few years, which has resulted in a devastating threat to the services provided by these companies. Figure 1.1, briefly shows the rise of DDoS attacks in recent years.

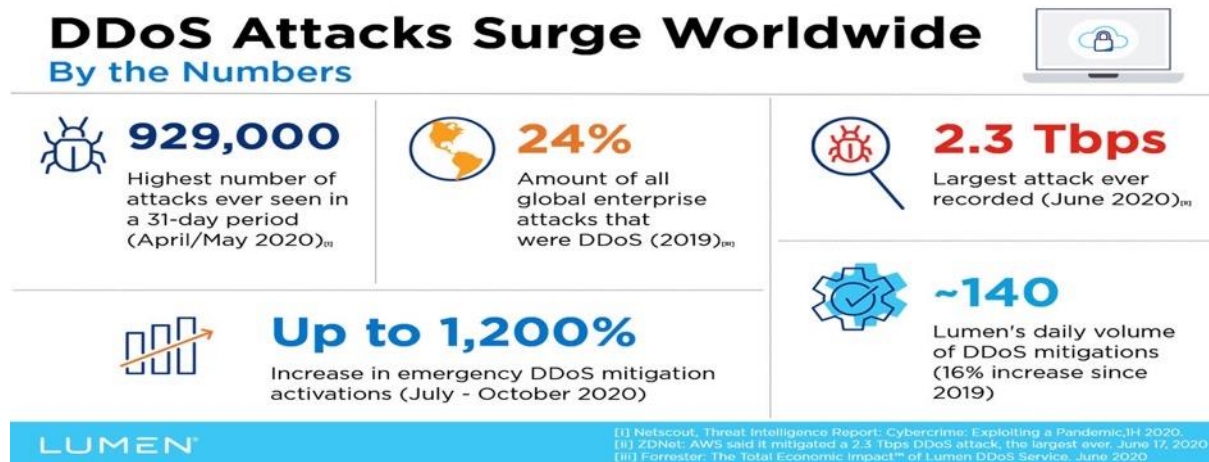


Figure 1. 1: The rise in sophisticated DDoS attacks in recent years [Taken from [6]].

Several instances indicate how DDoS attacks can affect the performance of different famous companies in recent years. One of the first major DDoS attacks targeted Yahoo! company, which brought down their services for a few hours [7]. In October 2002, another immense DDoS flooding attack took place, making the service unavailable for the majority of DNS root servers (9 out of 13) [8]. At the end of the year 2010, another harmful DDoS attack occurred when a group of hackers orchestrated massive DDoS flooding strikes on Mastercard, Visa, and PostFinance websites, bringing them down [9]. Attackers have lately targeted online banking websites, including several major US financial organizations such as Wells Fargo, PNC, HSBC, Bank of America, Citigroup, and U.S. Bancorp, which have all been subjected to massive DDoS flooding attacks [10].

Recent distributed denial of service (DDoS) attacks have become more complicated, and the scale of the attack has crossed the barrier of hundreds of gigabits per second (Gbps) [11]. One of the widespread denials of service attacks happened on October 21st, 2016 at Dyn. Dyn is an Internet infrastructure company that offers DNS service to a vast area of Europe and North America. In this attack, IoT devices were exploited to execute a Distributed Denial of Service attack, which created nasty DNS lookup queries [12]. This attack with a rate of 1.2 Tbsp., was the most devastating of its kind in the history of attacks [13].

1.1 Challenges in SDN network security

Security is an essential concern in both traditional networks and SDN networks since it ensures the integrity, privacy, and availability of information. The effectiveness of SDN is increasing in the IT world day by day; it spreads in diverse areas, from local area networks to public cloud architectures. However, despite the many benefits it offers, SDN security remains a matter of concern among research communities. SDN networks face a significant challenge in the form of the adverse effects caused by Distributed Denial of Service (DDoS) attacks.

It is noteworthy that the distributed denial-of-service attacks on the SDN networks can be much more destructive since the SDN architecture is based on the separation of the control plane from the data plane, where the intelligence of the network is centralized in a single entity known as the controller. The centralized role of the controller makes it vulnerable to DDoS attacks. This kind of attack can cause the interruption of the service of the entire

network by bringing down the controller [14]. As a result of the large number of unknown flows arriving at the controller, the controller's resources such as bandwidth, memory, and CPU, etc., become vulnerable, and the network becomes unusable. Furthermore, the bandwidth of the communication line between the controller and data plane can be saturated by a huge amount of this new traffic which will harm the functionality of the switches and disrupt the SDN network functionality [15]. Since the flow table is placed in the network devices, the data plane is also vulnerable to DDoS attacks. During the launch of the DDoS attack, enormous packets are sent to the network switches from diverse anonymous sources. The defined rules for these arriving packets will be added to the flow table of the switches by the SDN controller. The capacity of these flow tables is limited; therefore, the flow table of the switch will be saturated after a while. As a result, no new rules can be added to the flow table, and no packets can be forwarded. Furthermore, once the buffer capacity is saturated due to the DDoS attack, all incoming network traffic drops automatically [16]. Therefore, in recent years, the security of the SDN network came to the sight of the research communities and many researchers have tried to propose various methods to detect and mitigate DDoS attacks in SDN networks. They utilize various concepts and methods, such as statistical measurement, big data tools, and machine learning techniques to find a comprehensive solution to tackle this critical security concern in the SDN environment.

Adding big data analytics to a company's security architecture can significantly improve its capability to identify, investigate, and potentially prevent DDoS attacks. Large amounts of data must be analyzed and saved to evaluate DDoS attacks properly. The big data ecosystem is capable of managing massive volumes of data for assessment and comparison. Big data analytics provide comprehensive investigation, which can offer a vision into a diversity of crucial information, whereas standard traditional security methods contain only negligible logging information and just a few summary reports. It almost seems like hackers are able to modify their methodologies to include advanced IT concepts such as cloud capability prior to enterprises switching from scale-up to scale-out architecture. As a result, even the most sophisticated security systems are unable to detect, prevent, or mitigate today's DDoS attacks. In this situation, big data is a notion that can give companies knowledge about DDoS attacks and provide the tools they need to efficiently tackle the problem [17].

Big data analytics tools can provide various capabilities to address advanced and complex security threats. For instance, providing real-time data processing which is a significant capability for real-time threat analysis. Moreover, monitoring and automation of the network traffic offer the ability to react rapidly to detected threats. In addition, these analytical capabilities can be enhanced by using the machine learning approach. Using machine learning techniques in the network security domain and specifically for detecting DDoS attacks can assist researchers to develop a defense mechanism by collecting, analyzing, and processing data based on previous DDoS attack information. In other words, using machine learning algorithms enables a security framework to detect malicious traffic from activities learned and collected from past knowledge rather than through personal interaction, which in the first case it provides an automated, more accurate, and reliable defense solution.

in the realm of security, where novel threats emerge in less time and attack techniques are getting more sophisticated, it is vital to have formidable tools to facilitate manage and analyze all this new information to detect potential attacks swiftly and in real time.

1.2 Motivation

The rapid expansion of internet-based services in recent years has significantly increased cyber security concerns. Security in a data network should be a top concern since failing to do so exposes companies and organizations to a series of threats that might jeopardize service availability. The amount of data produced each day exceeds the threshold of petabytes. These data contain information about internet users' behaviors. On the other hand, besides the volume of data, the number of security threats is growing drastically. There are a wide variety of attack scenarios and methodologies, which makes them complicated and increases the difficulty of detecting them.

Different attacks have their own specific features to be recognized; hence a wide variety of signs must be considered to identify multiple attacks while monitoring a large-scale network infrastructure. DDoS attacks are one of the most significant issues in network security today due to the massive disruption they can cause to any type of network infrastructure [18]. The majority of distributed denial of service attacks are directed at online marketplaces, blogs, and the financial industry. It is also common knowledge that the SDN infrastructure is susceptible to attacks of this kind.

Software-Defined Networking (or SDN) is a networking technology that offers a particular type of network architecture, compared to traditional networks, which are more scalable and flexible. This modern architecture is also capable of reacting fast to shifting requirements for businesses and end-users' perspectives because of its specific characteristic, which is a unified management network [19]. The efficacy of software-defined networking (SDN) is becoming more and more apparent on a daily basis; its use spans a wide range of domains, from private local area networks to public cloud infrastructures. SDN demonstrates its excellent performance in the majority of situations by delivering simplicity, flexibility, reliability, and efficacy at a lesser cost. SDN security, however, continues to be a topic of concern among research communities, despite the numerous advantages of this technology. Similar to the traditional network, SDN-based networks also have the same security necessities [20].

The software-defined networking (SDN) viewpoint is predicated on the principle of separating the control plane from the data plane. In this architecture, all the intelligence of the network is unified in a single location, referred to as the controller. Due to the fact that in the initial design of the SDN architecture, security was not considered, different SDN layers are vulnerable to different attacks; figure 1.2 presents the effect of the DDoS Attack on the different layers of the SDN environment. Launching a DDoS attack targeting the SDN network can disrupt the functioning of the different SDN entities, such as overloading the Flow table of the network Switch, saturating the buffer of the switch, congestion of the southbound interface, and bringing down the SDN controller. The centralized controller, as the core of the SDN network, is susceptible to DDoS flood attacks, which may result in an interruption of service over the whole network [21].

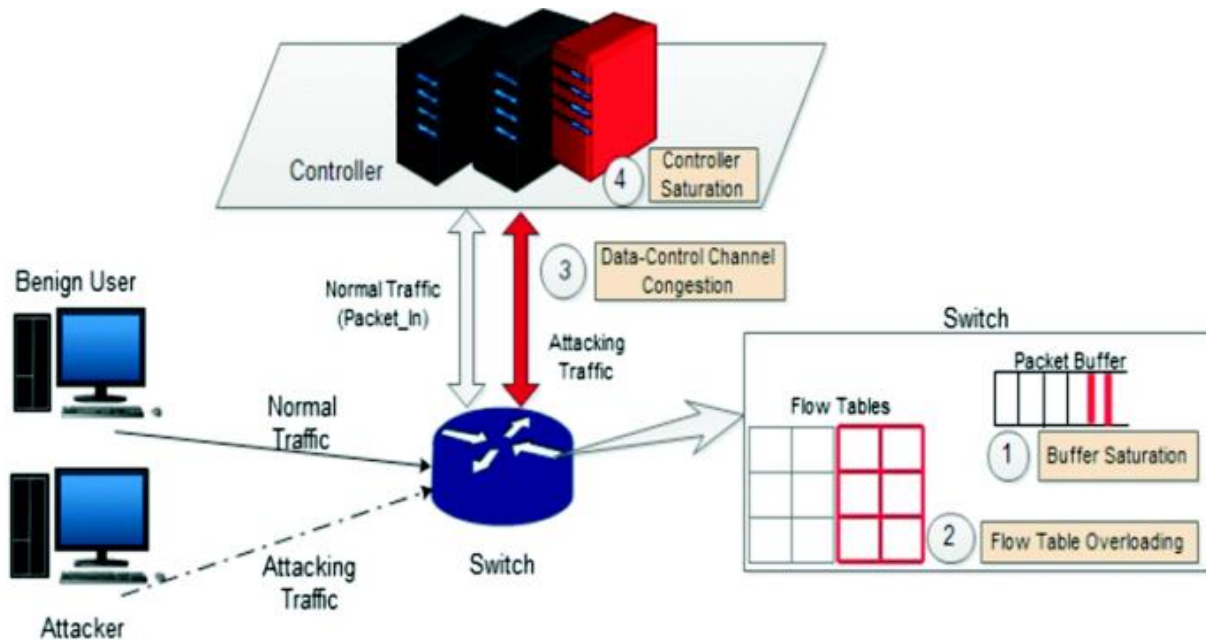


Figure 1. 2: Attack on SDN Layers. [22]

In the SDN-based network, the impact of distributed denial of service (DDoS) attacks on SDN networks is one of the most significant concerns of network administrators. A Distributed Denial of Service attack (DDoS) conducted against the SDN controller might deplete its processing capabilities, rendering it unreachable to legitimate traffic, thus affecting the availability of services [23]. In addition, because the controller will have all its resources dedicated to processing a large number of malicious packets, it will be inaccessible to legitimate traffic as a result of these types of threats. Driver functions, such as online services or web applications, may become unavailable because of these types of threats [24].

Compared to conventional networks, the effectiveness of distributed denial of service attacks (DDoS) will be evident in SDN networks at a much quicker pace and with higher damage. There are now several research papers that offer various techniques for detecting and mitigating distributed denial of service (DDoS) attacks in SDN networks. However, there is no methodology that serves as a guide for the implementation of these solutions [25].

In most of these proposed methods, the module is centrally located in the controller. There are three major disadvantages that can be addressed for these methods:

- 1- One of the main drawbacks to these techniques is the lack of scalability of the detection module; In most proposed methods the detection/defense module is implemented by developing the controller core or running it as an application in a separate machine, in both cases, it can be a single point of failure.
- 2- In centralized detection methods, concerns are raised about the amount of computing load imposed on the controller during heavy DDoS attacks. When a DDoS attack begins, a large number of new packets flow to the controller. The controller must analyze all new packets and take the essential actions. In this case, a considerable computing load is imposed on the controller, and the controller may fail to manage and control other upcoming network requests.

- 3- Another issue is the delay in responding to permissible flows sent to the controller. The validation process for detecting malicious traffic to any new incoming network packets by the controller can cause delays in responding to legitimate flows. For the SDN network, these additional delays can be a major concern.

Therefore, with the increasing complexity of the different DDoS attack techniques for deploying the SDN architecture in a network environment, it is crucial to consider the above-mentioned concerns. To protect the SDN network, sophisticated methods are essential to monitor, analyze, and accurately identify attack features. In this context, big data techniques can be used to detect these attack features. An edge might be gained against these kinds of attacks by developing a real-time network traffic monitoring and processing system that is able to identify malicious activity, scale the amount of data that is ingested, and react quickly in terms of reaction time. To ingest and analyze massive volumes of data in real-time with minimal latency, distributed processing systems such as Apache Spark ¹, Apache Kafka², and Apache Storm³ are rapidly being deployed. Distributed processing solutions, such as Apache Kafka, and Apache Spark are being quickly implemented so that huge amounts of data may be ingested and analyzed in real-time with little lag.

Using these big data tools, a DDoS detection system may quickly detect abnormal activities or suspicious acts and mark them for urgent analysis. This can be done during enormous workloads with heavy traffic. Developing such a framework can offer a vital supporter that may assist network security systems in addressing the DDoS threats that are currently being encountered in the SDN environment. In this sense, machine learning (ML) algorithms provide a viable opportunity for classifying network data and determining attacks.

Machine learning approaches can analyze characteristics and learn from them, which enables the detection and prevention of DDoS attacks. In the context of DDoS attack detection, approaches based on machine learning perform very well when combined with the robust qualities of big data processing systems.

In general, in the context of the DDoS attack detection methods, there are still certain open issues which are summarized in the following:

- Designing a real-time attack detection system to collect and analyze incoming new packets without packet loss.
- Implementing security techniques to the SDN controller may influence the performance of the entire network during high traffic load.
- To prevent controller performance degradation, the detection system should be decoupled from the core network architecture.
- The attack detection system should be continuously updating its attack features, information, and detection methodologies to be capable of detecting new attacks without

¹ "Apache Spark". [Online]. Available: <https://spark.apache.org/>, /, [Last accessed: June 2022].

² "Apache Kafka". [Online]. Available: <https://kafka.apache.org/>, /, [Last accessed: June 2022].

³ "Apache Storm". [Online]. Available: <https://storm.apache.org/>, /, [Last accessed: June 2022].

jeopardizing the system performance, as attack scenarios and patterns change very fast and attack methods are getting more complex and difficult to detect.

To tackle the above-mentioned issues, the big data analytics tools with their specific powerful features such as the formidable processing power, real-time analysis, and capability of handling heavy traffic can be used to design and develop a DDoS attack detection and prevention framework for the SDN network. To address the issues mentioned in the SDN environment, we propose a framework that utilizes machine learning techniques and a big data pipeline. This framework provides rapid, precise, efficient, and automated security systems that can identify DDoS attacks in real-time. Additionally, it can be easily adapted to new malicious activities and can handle large traffic loads

1.3 Objectives

The DDoS attack is still a severe threat to SDN networks, where there exists no specific solution that provides a perfect security orchestration against DDoS attacks in SDN networks. It is crucial to have a solution that can quickly and reliably identify DDoS attacks in real-time network traffic. The security solution must be fast, scalable, and effective to provide the SDN network environment efficiency and security. In this research, since the control plane is the most vulnerable part of the SDN security architecture, we specifically consider and investigate the control plane of the SDN architecture to design and develop a methodology prototype for detecting and mitigating the DDoS attack directed at the control plane and southbound interface in the SDN architecture with the aim of improving the security within SDN network. We propose a scalable, reliable framework to detect and mitigate DDoS attacks to prevent the computational load and delay in the legitimate flows sent to the controller.

1.3.1 Research Questions

In this research, we intend to develop a methodology to detect and mitigate DDoS attacks using machine learning techniques based on big data pipelines to increase security in the control plane of the SDN environment. Therefore, it is critical to design a software stack in a way that the collected data is sent in real-time for analysis by the model.

there are the following major questions to be investigated in this research:

- 1- What are the SDN attack vectors?
- 2- Which machine learning classifier algorithm is appropriate for detecting DDoS attacks?
- 3- How to implement a data pipeline to detect malicious network traffic in the networks?
- 4- How can a detection module be integrated into the SDN controller for the detection and prevention of a DDoS attack in the SDN environment?
- 5- What is the effect of integrating a detection module using the big data pipeline and developed controller to detect and mitigate DDoS attacks in software-defined networks?

1.3.2 Specific objectives

The overall objective of this research is to develop a framework that is a combination of a data analytics pipeline for processing the incoming traffic and identifying network abnormalities in real-time. This will be accomplished by developing a data analytics pipeline. To be more specific, the following four activities will be incorporated into the development of the framework:

Activity1: Problem identification and motivation

Activity2: Development of the Control Plane of the SDN network

Activity3: Evaluation of Different Machine Learning Classification Algorithms

Activity4: Implementation of Big Data Pipeline for Intrusion Detection

Activity5: Demonstration of the Framework with an experimental setup

Activity6: Evaluation

In the following, each activity is described more precisely:

Activity1: Problem identification and motivation

-Investigating the literature associated with software-defined networks and their applicability in this research.

- Researching SDN security issues and defining various security vectors and possible solutions on different layers of the SDN architecture.

-Studying the different types of DDoS attacks in SDN and their detection and mitigation mechanisms.

Activity2: Development of the Control Plane of the SDN network

-Developing an API into the SDN controller to handle, aggregate, and convert the incoming traffic information immediately after they arrive and make them reliably accessible for the rest of the pipeline.

Activity3: Evaluation of Different Machine Learning Classification Algorithms

-Selecting the most appropriate machine learning algorithms for the identification of DDoS attacks.

Activity4: Implementation of Big Data Pipeline for Intrusion Detection

-Carrying out a study of big data and technologies and architectures for massive data analysis.

-Designing the methodology based on the big data pipeline for the detection and mitigation to identify, analyze, and prevent the different denial of service attacks in the SDN network.

-Designing and building classification models using big data tools capable of classifying network traffic and detecting possible attacks with good accuracy and a low percentage of false alarms (the classification of normal traffic and attacks). Implement a machine learning technique that processes network information into big data tools and makes predictive analyses with the most recent dataset to date.

Activity5: Demonstration of the Framework with an experimental setup

-Demonstrating the proposed framework by integrating the traffic capture tool, developed SDN controller, and the machine learning model with the big data software stack to obtain a distributed intrusion detection system capable of analyzing and classifying the flooding traffic targeting the victim system in real-time.

Activity6: Evaluation

-Evaluating and verifying the viability of the proposed detection and prevention methodology in a controlled simulation scenario using real attacks and checking the results.

-Drawing conclusions from the analysis and proposing improvements and visualizing the results of the monitoring process.

To accomplish each individual objective of our project, we have used a unique collection of tools to address each challenge with the appropriate approach.

1.4 Goals and Contributions

To deliver an accurate, scalable, real-time, and fast DDoS detection and prevention system that matches current demand, our solution integrates SDN concepts, machine learning, and distributed stream processing using a big data pipeline. The challenging task in this research is to develop a machine learning-based network intrusion detector using big data tools, a predictive framework that is capable of distinguishing between malicious traffic and legitimate traffic.

More specifically, in the primary phase of the proposed solution, a Java-based API is developed into the SDN controller for collecting, aggregating, and transforming the network traffic information received from the controller into a specific format. These collected data are constantly sent to a distributed message broker known as Apache Kafka, whose job is to provide a fault-tolerant layer for the data stream processing phase. It consistently saves and manages the input network traffic information, which is sent via the SDN controller to feed them to the data stream processing step of the big data pipeline.

The next step is where the majority of the processing and classification of the network traffic is done. It is implemented using the Apache Spark framework, which is a scalable analytics tool for data processing data and includes several different packages for various purposes such

as Structured Streaming and machine learning. For classification of the network traffic, first pre-processing should be done to boost the accuracy by removing unrelated features. Afterward, five different supervised classification algorithms, such as Logistic Regression, Naïve Bayes, k-Nearest Neighbor, Random Forest, and Decision Tree are evaluated to find out which algorithm has the most efficiency in differentiating between normal and malicious traffic.

In this research, to evaluate the deployed machine learning algorithm which is implemented in the proposed framework, we have decided to use the NSL-KDD [26], and UNSW-NB15 [27] datasets. The NSL-KDD and UNSW-NB15 datasets are popular datasets that are extensively used as a benchmark for contemporary network traffic supervising systems. The last step of the data pipeline consists of storing the classified network flows. For this function, we use Elasticsearch⁴, a distributed search engine, to provide a fast search and store service. When the network flows are processed through the data stream processing step of the big data pipeline, if any malicious flow is detected, a request will be sent to the SDN controller via the REST API communication channel to block the sender of the traffic.

In another attack scenario, the attacker is targeting the southbound interface to saturate the channel between the controller and the network infrastructure devices. In this situation, the controller cannot receive the network traffic to perform the data processing for distinguishing the malicious traffic. In the proposed approach, we can tackle this serious vulnerability by setting up the infrastructure devices to send the network flow information via the standard network protocols such as Netconf and IPFIX to the data processing pipeline, and the rest of the process will be the same. The network devices such as switches can be configured to send the network traffic information via Netconf or IPFIX protocols to the big data pipeline, and if any malicious traffic is detected, the sender will be blocked by sending a REST communication request to the controller.

1.4.1 Thesis Impact

During this research, the result of different parts of this thesis have been published in peer-reviewed conference proceedings:

- 1- Amirreza Fazely Hamedani, Muzzamil Aziz, Phillip Wieder, Ramin Yahyapour, " Security Enhancement in SDN-based Networks Using Big Data Analytic Pipeline.", *Digital Privacy and Security Conference 2023 (DPSC2023)*.

Own Contribution

I am the lead author of the paper. All major analyses and evaluations have been done by myself.

⁴ "Elasticsearch. Distributed search and analytics engine based on Apache Lucene." [Online], Available: <https://www.elastic.co/>, [Last accessed: June 2022].

- 2- Amirreza Fazely Hamedani, Muzzamil Aziz, Phillip Wieder, Ramin Yahyapour, "Leveraging Big Data Pipeline to Enhance Security in SDN networks.", The 27th International Conference on Optical Network Design and Modelling (ONDM 2023), Potugues.

Own Contribution

I am the lead author of the paper. I have to present the investigation and evaluation of the proposed method in the workshop.

- 3- Amirreza Fazely Hamedani, Muzzamil Aziz, Phillip Wieder, Ramin Yahyapour," Big Data Framework to Detect and Mitigate DDoS Attacks in SDN Networks", Proc. of the International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME 2023), Spain.

Own Contribution

I am the lead author of the paper. I significantly contribute to designing and implementing the prototype model.

- 4- Amirreza Fazely, "DDoS attack and Detection Techniques", GWDG-Nachrichten, March 2019.

Own Contribution

I am the lead author of the paper. I did the most investigating and writing.

Furthermore, one papers have been prepared for submission to the IEEE conference:

- 5- Amirreza Fazely Hamedani, Muzzamil Aziz, Philipp Wieder, Ramin Yahyapour," Analysing the Efficiency of Utilizing Machine Learning Techniques on Apache Spark for Intrusion Detection", 4th International Conference on Digital Technologies and Applications,ICDTA'24, February 2024,Fez, Morocco.

Own Contribution

I am the lead author of this paper. As the primary researcher, I took on the responsibility of thoroughly performing the analysis and evaluation of the different machine learning approaches discussed in this study. This hands-on involvement ensured the accuracy and reliability of the results presented in the paper.

Moreover, a few papers were published to which the author of this thesis contributed:

- 6- Muzzamil Aziz; Amirreza Fazely; Giada Landi; Domenico Gallico; Kostas Christodoulopoulos; Philipp Wieder," SDN-enabled application-aware networking for data center networks", *IEEE International Conference on Electronics (ICECS), 2016.*

Own Contribution

My own contribution to this paper is the implementation and modeling of the required simulation prototype. Furthermore, performing the evaluation and analysis of the simulated data.

- 7- Giada Landi; Ioannis Patronas; Kostas Kontodimas; Muzzamil Aziz; Kostas Christodoulopoulos, Amirreza Fazely, "SDN Control Framework with Dynamic Resource Assignment for Slotted Optical Datacenter Networks", *Optical Fiber Communication Conference, OSA Technical Digest, 2017*

Own Contribution

Own contributions to this paper comprise designing and implementing the proposed approach.

- 8- Amjad Zia, Muzzamil Aziz, Ioana Popa, Sabih Ahmed Khan, Amirreza Fazely Hamedani, Abdul R Asif, "Artificial Intelligence-Based Medical Data Mining", *Journal of Personalized Medicine, 2022.*

Own Contribution

Own contribution to this journal paper embraces certain conceptual investigations and evaluations of various machine-learning techniques for medical data mining.

1.5 Thesis Outline

The rest of the report is organized as follows:

In Chapter 2, we provide a review of the current literature on the different defense methodologies. We divided the existing literature into five categories: General Methods, Mathematical Methods, Methods, Big Data Methods, and Hybrid approaches. Furthermore, there is a detailed review of the current literature for each category.

In Chapter 3, we explain an overview of the SDN network, its history, the concept from the perspective of general definition, and its particular characteristic, which separates it from traditional technologies. In addition, we introduce the details of the SDN architecture and its different layers from top to bottom point of view. Furthermore, we present the OpenFlow protocol and its specification, since it is the protocol that we consider in our proposed framework prototype as a southbound interface for communicating between SDN controllers and infrastructure switches. For this reason, we describe the OpenFlow architecture and OpenFlow-based switches, followed by its workflow in terms of how messages are exchanged between the controller and the switches and different types of messages.

Then, we summarized the SDN security and different challenges and issues that should be considered. More specifically, in section 3.6, we explain the Threats and SDN vulnerabilities related to different layers of the SDN architecture. In section 3.6.3, we expound on the seven significant SDN attack vectors. Section 3.7 introduces the Denial of Service (DoS) attack, including its specification and why it has become an important subject for security researchers and organizations. Particularly, in section 3.7.2, we clarify the effect and impact of DDoS attacks on three layers of the SDN structure.

In Chapter 4, initially, we introduce various key factors which are essential for evaluating different SDN controllers; then we investigate in detail five prevalent SDN controllers based on these key factors in order to choose the proper controller to be implemented into our

framework. For this purpose, we particularly study various prevalent SDN controllers, including NOX/POX, Ryo, Floodlight, OpenDaylight, and ONOS, and illustrate their overall architecture.

In Chapter 5, we explain the proposed framework and its components and modules. Particularly, in sections 5.1 and 5.2, we describe the details of the proposed framework architecture, its three major modules, its functionality, and workflow. Section 5.3, contains information about developing an API for the SDN controller. Evaluation of the different algorithms and the results are explained in Sections 5.4 and 5.5. In section 5.4, we provide a theoretical explanation of, its use cases, and three major classifications. This is followed by defining the process of designing and evaluating the accuracy of a model. Particularly, Section 5.4.5, contains the mathematical background of various algorithms which we use in our research. Further, in section 5.5, we depict the process of selection of the proper algorithm in detail and present the comparison result of the selected classification algorithms.

Furthermore, in section 5.6, we provide information about the tools and technologies used in this research. Section 5.6.2, contains details about the big data tools, including, Apache Spark, Spark Streaming, Apache Kafka, Zookeeper, and Elasticsearch, which we employ to implement the data pipeline for processing the network traffic to detect DDoS attacks. The details of the big data pipeline implementation are described deeply in section 5.7.

In Chapter 6, we describe an experimental setup and performance evaluation. Specifically, section 6.1 contains the details about the experimental testbed and its specifications. In sections 6.2 and 6.3 we present the deployment of the network topology and proposed framework operation. Sections 6.4 and 6.5 present experimental results and performance evaluation.

In Chapter 7, we introduce three European projects that we include as contributions and use-cases of this research to present how the SDN paradigm can provide a possibility to modify the network infrastructure on demand to match the needs of any organization.

In Chapter 8, we summarize this research and briefly explain the achievements and conclusions. Finally, some possibilities for the development of the future plan for this thesis are discussed.

Chapter 2

2 Literature Review

In the last few years, due to the significance of security risks in software-defined networks, different detection and mitigation systems against DDoS attacks have been proposed in the literature; Numerous concerns exist in the traditional detection system such as no real-time reaction, slow detection process, single point of the failure, etc. Based on our knowledge, few researchers offered a distributed processing framework for DDoS detection and mitigation system in SDN networks to address the issues of traditional systems. Most of them rely on techniques such as methods, statistical solutions, and information theory. In some statistical solutions, entropy has been used to detect the attack [28].

Although entropy is a proper analytical method, it alone cannot detect all the attacks. Therefore, for some attacks, it is probable that the system may trigger false alarms at peak times. The peak time is when the legitimate traffic on the network has been increased for some reason, such as providing authorized services to too many users. Therefore, the best approach is to combine entropy with some other methods. In some other solutions, researchers have offered graph-based methodologies for the detection of DDoS attacks. However, there also lies the possibility of false alarms with the increase in illegitimate traffic [29].

2.1 General Methods

Kandoi et al. examined the risk of controller resource saturation and flow table overflow. This paper outlines a method that revolves on altering and modifying the configuration of switches to deal with these threats [30]. Taejin et al. introduced multiple intrusion detection systems. In this method, the traffic flow is distributed among the systems using the flow group clustering algorithm based on routing information and data flow rate [31].

Wang et al. proposed a mechanism of protection against Distributed Denial of Service attacks in cloud environments using the Floodlight controller. This method consists of two modules, one for detection and the other for defense. The attack detection module is responsible for identifying attacks and then notifying the response module for performing the necessary action against the attacks. This method is the first solution against this type of attack in cloud environments with the help of SDN [32].

Fichera et al. offered a technique for protecting web servers from Flood attacks based on the Pox controller. This technique introduces a controller named OPERETTA, which verifies each SYN packet and then allows the connection to start. This method is very comparable to SynCookie on traditional networks [33].

Discussion:

These general DDoS detection methods involved enforcing specific network policies or rules on the attacked network as traffic was flowing through it. It is also possible to define various network policies and traffic rules to prevent and identify various DDoS attacks.

On the other hand, these solutions have some limitations, including the fact that it only applies to a single SDN controller and does not support scalability and also real-time detection.

2.2 Mathematical Methods

as a mathematical method, Chouhan and Peddoju use the hop count of the IP mapping table for clarifying the DDoS attack packets. They claim that the accuracy for detection in this technique for IP attack behaviour, has been around 90% and deployment of the method is comfy. As a negative point, this technique is helpless against distributed attacks. Furthermore, if the IP-to-Hop-Count mapping table (Known as P2HC) is not updated appropriately, normal network packets may be mistakenly considered malicious traffic which affects the false positives [34]. As another mathematical method, Feinstein *et al.* consider the entropy and chi-square assessment for the network traffic to detect DDoS attacks [35].

Discussion:

In addition to the aforementioned mathematical methods, there are various other similar options available for SDN networks. In these methods, probability, the analysis of standard deviation, and correlation. and entropy measurement, are the essential foundation. For instance, Entropy can be calculated by considering two different variables: the specifying threshold value and the clarifying window size.

As main drawback of these methods is, when the network traffic complies with the traffic regulations and policies, the network traffic is considered to be legitimate traffic flows; otherwise, in the vast volume of the traffic, even the legitimate traffic, they are regarded as malicious traffic flows. In addition, these methods, do not offer any platform or framework that can be implemented in the real world.

2.3 Machine Learning Methods

In this research, they use three distinct kinds of data mining applications to comprehend the prototype and utilize them for DDoS attack detection, in order to assess the efficiency of the proposed protection architecture. Their experience shows a proper outcome on attack detection and false positive rate.

Gurulakshmi and Nesarani utilize the classification method and specifically the SVM method to classify normal and malicious traffic and predict immediate abnormal activities [36]. Moreover, Huang *et al.* suggested architecture for network security based on Deep defense, and also proposed data mining techniques for processing and evaluating the alarms collected in the IDS/IPS systems [37].

In another research and based on the C4.5 algorithm, Zekri *et al* offered a DDoS attack detection. In their proposed method the signature detection technology is considered to create the decision tree, and the algorithm is used to detect the DDoS flood attack automatically and efficiently [38]. Wang *et al.* proposed a technique based on flow mining for attack detection, by using the Code Red and Slammer worms and DDoS data collection from MIT Lincoln Laboratories, they indicate the effectiveness of this approach [39].

Rojalina et al. [16] developed a defense system and detection methods against DDoS attacks based on deep learning. In this method, the capabilities of SDN networks have been used to design a prevention algorithm module in the SDN controller. Robust infrastructure requirements and high costs are some of the problems associated with this method. The accuracy of the system in detecting attacks is reported to be 98.88% [40].

Discussion:

The major vulnerability of the SDN network is the centralized controller. This is due to the fact that in the SDN architecture, the underlying infrastructure which is called the data plane, and control logic which is called the control plane have been separated. To protect SDN controllers against DDoS attacks, there are already several methods. The machine learning approaches are able to analyse network traffic characteristics and learn from them, which can well assist in the detection of DDoS attacks.

One of the disadvantages of these methods is, they cannot provide any scalability and reliability for their methods which is a crucial feature for any network attack detection method.

2.4 Big Data Methods

G. Dileep Kumar et.al introduced a real-time DDoS attack detection method based on SDN and big data. In their study, they implemented an SVM algorithm and offered two novel approaches for attack detection. They claimed that the proposed method could further be improved to detect and prevent different types of DDoS attacks [41]. Zhao et al. developed a neural network for detecting DDoS attacks which has the ability to adapt to new types of DDoS attacks. In their proposed system, they used big data tools such as HBase and Hadoop for analysing and storing huge amounts of the unstructured dataset which have been collected from the network log. [42].

Sufian and Usman presented a system called HADEC, which is a live DDoS detection framework based on Hadoop to deal with flooding attacks using MapReduce. They implemented an algorithm that only detects the TCP-SYN, ICMP, UDP, and HTTP GET attacks [43].

Karimi, et al., by using Spark offer a feature extraction technique for real-time IDS. For capturing the network packets from the network devices and obtaining essential information from the packet header, they utilize a collector module. All this captured header information is stored in CSV files and used the time window to separate them. Then, for providing real-time processing, Spark periodically uses these CSV files for reading data during a small-time window. One disadvantage of this method for the online Internet traffic monitoring system is

that reading and writing the data from the CSV files in an intervallic way may reduce the Spark performance [44].

Gupta et al. proposed three network traffic monitoring tools by using the Spark Streaming library for analyzing network packets in real-time. This application, which involves monitoring reflection attacks, may be described as streaming analytics problems, port scan detection, and application functioning evaluation. To obtain only the traffic that is required, they utilized the OpenFlow switches (programmable switches). The advantage of this solution would be reducing the amount of data that should be processed. On the other hand, this methodology is not realistic for a network that utilizes traditional switches (non-programmable switches) [45].

Lee et al introduced a method based on Hadoop⁵ for HTTP flooding attack detection. In their approach, the Hadoop-based packet processor [46] is used and developed a detection method using MapReduce⁶ to detect the HTTP GET flooding attack. In their approach, the number of the traffic volume or the total number of requests for the web page is calculated using a MapReduce method for detecting DDoS using counters to distinguish the attackers from legitimate users. For evaluating the performance of the proposed method for DDoS attack detection, multiple Hadoop nodes (max. 10) are used in parallel. There is an issue regarding the proposed framework. This technique in the current structure can only be used for offline batch processing of a massive number of requests. Therefore, developing this approach to provide a live analysis using a real-time defense system should be considered [47].

Discussion:

The approaches that have been suggested offer a mechanism that is quick for the identification of malicious network traffic. These solutions have some limitations, for instance since some of these methods can only operate at the transport layer, it is unable to detect network threats throughout the entire OSI network layers. On the other hand, these methodologies are not realistic for a network that utilizes traditional switches (non-programmable switches).

In addition, the methodology that some of these techniques use is based on Hadoop which is somewhat slow and inefficient when real-time processing is required. In fact, the speed of Apache Spark which we use for data processing in our proposed framework is much faster than Hadoop either running in memory or on a disk [48].

2.5 Hybrid Methods (Big Data and Machine Learning)

Saied et al. presented a method for attack detection using the ANN algorithm and using various big data tools, such as Apache Spark for a cluster computing system, HDFS for data storing, and Yarn for the resource management [49]. Dahiya et al. used Apache Spark to develop an intrusion detection system in NetFlow protocol. Linear Discriminant Analysis (LDA) and Canonical Correlation Analysis (CCA) were employed as two feature reduction approaches

⁵ "Hadoop". [Online]. <https://hadoop.apache.org/>. [Last Accessed July 2018]

⁶ "Mapreduce". [Online]. <http://wiki.apache.org/hadoop/MapReduce>. [Accessed July 2018]

and then applied supervised classification machine learning algorithms including Random Forest, REP Tree, Naive Bayes, and Bagging using UNSW-NB15 as the dataset [50].

Beluch et al. investigated the performance of various famous machine learning (ML) algorithms such as SVM, decision tree, Naïve Bayes, and Random Forest using Apache Spark. For the performance evaluation, they used UNSW NB15 as their dataset with 42 features in terms of accuracy, building time, and prediction time. They reached the conclusion that the Random Forest algorithm with 97% accuracy has more advantages compared to the other classifiers [51]. Ferhat et al. used the k-Means algorithm in the machine learning libraries on Apache Spark to decide between the attack or normal network traffic. They used the KDD Cup 1999 is used for training and testing and didn't consider any feature selection technique to select the related features [52].

N. Patil et al. developed S-DDoS, a distributed and real-time DDoS detection system based on Spark streaming, to categorize live traffic flows. To distinguish the DDoS attack traffic in real time, they employed the K-Means clustering algorithm. Also, by using highly scalable H2O sparkling water, they designed the model on the Apache Hadoop framework. They asserted that the suggested S-DDoS detection method accurately and successfully identifies DDoS attacks from network traffic flows (98%) [53]. Manzoor et. proposed a real-time intrusion detection system based on the SVM algorithm using the Apache Storm framework. For intrusion detection, the authors employed C-SVM with libSVM categorization. The proposed method was trained and assessed on KDD 99 dataset. The experimental findings demonstrate that the suggested approach is practical for stream processing of network traffic data for highly accurate network intrusion detection. [54].

Kai Peng et al. offered an IDS system built on a decision tree over big data in a Fog computing environment using the classification machine learning approach. To digitize the strings in the presumptive dataset, they implemented a pre-processing technique. They then normalized the data to ensure the accuracy of the input data and boost the effectiveness of detection. They compared the Nave Bayesian and KNN methods with the decision tree method for IDS. The results that they obtained using the KDDCUP99 dataset demonstrated that the suggested strategy is effective and precise. [55]. Zhao j *et al.* designed a methodology using a combination of big data analysis with machine learning, dynamic instruction flow analysis, and binary instrumentation to propose a technique for classifying the malware [56].

Trezi D. *et al.* utilize Principal Component Analysis (PCA) method for dimension reduction and offer unsupervised anomaly detection using Apache Spark. In addition, they indicate that for implementing big data for anomaly detection, multiple options such as Scalability, choosing appropriate features, and Confirmation of learned knowledge should be considered [57]. Alsirhani *et al.*, for anomaly detection, proposed multiple classification approaches in spark, and furthermore, for selecting the classification algorithms used Fuzzy Logic and demonstrated this method on MATLAB. Their experimental outcomes revealed that the tree-based classification algorithms have an enhanced classification influence on the traffic classification [58].

Ying Gao *et al.*, by using big data and distributed DDoS network intrusion detection, provide a new approach to network intrusion detection. A real-time traffic collecting module and a traffic detection module are the two main components of their system. In their framework

architecture, they used Apache Spark for fast data processing and HDFS for storing fishy networks. To enhance the real-time performance of the traffic feature collection, they adopted a micro-batch data processing paradigm in the network collecting component. Furthermore, they implemented the Random Forest classification algorithm in the traffic detection component. They also utilized two data sets, NSL-KDD and UNSW-NB15, to evaluate the accuracy of the framework in attack detection. The results of their evaluation using these two data sets indicate an accuracy rate of 99.95% and 98.75% for the proposed detection algorithm and a false alarm rate of 0.5% and 1.08% [59].

Tizghadam *et. al.* developed a system called CVST. It is an open-source, scalable manifesto that can be used in smart transport application development. Their proposed system contains four specific major modules for different use cases such as business intelligence, data dissemination, resource management, and application. For instance, the MLLib library is used in the business intelligence module, which is responsible for data analytics, to process the information and send it to the front-end [60]. Arora implemented the K-means machine learning algorithm on the Apache Spark MLLib library to perform an analysis of the mobile data which are collected from the internet. He proposed an efficient approach by using clustering based on latitude and longitude values for estimating the number of clients in the network [61].

Discussion:

These above-mentioned methods are the combination of multiple advanced methods to provide a scalable, fast, real-time platform for the detection and mitigation of DDoS attacks. Most of these methods offer a solution for a general network, which means that they may not be appropriate to be implemented in the SDN network, and they cannot take advantage of using a centralized controller to control, maintain, and manage the security of the entire network.

2.6 Research Gap

The identification and mitigation of Distributed Denial of Service (DDoS) attacks in Software-Defined Networking (SDN) networks is a critical research area. Although significant progress has been made in this field, there are still some research gaps that need to be addressed. In this section, several potential research gaps have been proposed that could be explored further:

- 1- **Lack of Standardization:** The field of SDN is still relatively new and lacks standardization. As a result, there is a lack of standardization in the detection and mitigation of DDoS attacks in SDN networks. Future research could explore ways to standardize the detection and mitigation process, which would make it easier to implement and scale across different network environments.
- 2- **Real-Time Detection:** Most existing approaches to detecting DDoS attacks in OpenFlow-based SDN networks rely on post-attack analysis. However, real-time detection is critical to minimizing the impact of DDoS attacks. Future research could explore ways to improve real-time detection by analyzing network traffic in real-time and using machine learning techniques to identify anomalous patterns.

- 3- **Low Negative Impact on Legitimate Traffic:** Additionally, research should focus on developing mitigation strategies that can effectively counter DDoS attacks in SDN networks, while minimizing the impact on legitimate network traffic.
- 4- **Scalability:** While scalability is one of the primary goals of OpenFlow-based SDN networks, existing approaches to detecting and mitigating DDoS attacks may not be scalable enough to handle large-scale attacks. Future research could explore ways to make DDoS detection and mitigation mechanisms more scalable, such as by using distributed detection techniques or optimizing the use of network resources.
- 5- **False Positive and False Negative Rates:** The effectiveness of current DDoS detection and mitigation mechanism in SDN network depends on its ability to minimize false positive and false negative rates. However, existing approaches may suffer from high rates of false positives or negatives. Future research could explore ways to improve the accuracy of DDoS detection and mitigation mechanisms, such as by using machine learning techniques or analyzing network traffic at multiple levels of granularity.
- 6- **Cost-Effectiveness:** Implementing effective DDoS detection and mitigation mechanisms can be costly, both in terms of time and resources. Future research could explore ways to make DDoS detection and mitigation mechanisms more cost-effective, such as by optimizing the use of network resources or using open-source tools.

In conclusion, the detection and mitigation of DDoS attacks in SDN networks is an important research area that requires further investigation. The proposed research gaps provide a starting point for researchers to advance the current state of knowledge in this field.

2.7 Chapter Summary

The majority of this chapter is devoted to a literature review that examines security concerns associated with SDN. Defense methods against DDoS attacks that currently exist were addressed. These mechanisms were distributed into five different sections. The first one is the “General Method”, then the second one is the “Mathematical Method”. We consider these two methods as traditional methods. Then we alternatively study the “Machine Learning Methods”, “Big Data Methods”, and “Hybrid Methods” as the most recent methods for detecting and mitigating DDoS attacks. Furthermore, there has been a detailed review of the current literature for each section. These investigations provide us with the essential knowledge to identify the research gaps that will contribute to the accomplishment of the proposed research objectives.

Chapter 3

3 Technology Background

SDN is a revolutionary network architecture that stands for software-defined networking. The concepts proposed by SDN, such as separation of the control and forwarding planes, centralized control of network status, and support for software programming, are nothing new, but there has been no breakthrough for a long time. Currently, because of the IT's conversion in network demand, SDN has attracted pervasively IT experts' attention: cloud computing services (represented by server virtualization technology) have become conventional, big data technology has become increasingly popular, and network resources are elastically expanded. The weaknesses of the traditional model are that in that model, the network devices are closed systems that consist of three components: the hardware, the operating system, and the network application. These components are tightly integrated with one another. The development and invention of each of these three components necessitate the evolution and innovation of the others.

The last decade has endorsed an extraordinary revolution in end-user devices both in terms of computing capacity and in terms of intelligence level. The presence of a large number of smart devices, high-performance tablets, phones, and servers is a clear example of the computing power that has been provided for users. Today's network infrastructure carries much more data over networks for connecting a larger number of people, more complex applications, and devices, which demands improvements in the processing capability of the network hardware and provides enhanced control over data traffic. One solution to overcome the above-mentioned concern is to replace existing processors which are used in the network equipment with faster ones, but due to the strong relationship that exists between the architecture of the processors and the software that operates on them, this solution is not feasible.

Upgrading and replacing the current software, as another possible solution, is also not a viable option since the infrastructure of the networks is made up of equipment from different manufacturers that must interoperate with each other and require great efforts to achieve this compatibility. Above mentioned condition cause limitations on any improvement and innovation in the conventional network and due to this limitation and to offer a potential solution the following questions arise:

Is it possible that the algorithms and computation part to be moved from network equipment? Can control part of the network equipment be moved to a central entity and, also provide programmability to the network? Is it feasible to create a more open, flexible, and intelligent network? The answer to all these questions guided the researchers to replacement of the traditional network model with the software-defined network model, where the control plane of the network equipment is moved to a logically centralized point and, also provides the

capability of programmatically controlling the forwarding network traffic for many network devices [62].

3.1 Overview

The computer network, in general, is a set of heterogeneous equipment (switches, routers, computers, etc.) that is interconnected with each other to exchange information. There are standard models which allow different network devices to interact and exchange their information. The two major structure models used for the planning and implementation of networks are the Open System Interconnection (OSI) model [63] and TCP / IP [64]. The model OSI is a network communication standard developed by the International Organization for Standardization (ISO) in 1984, specifying how network equipment should communicate with each other. Its architecture is composed of seven layers, and each layer is independent of the others to perform a particular role.

comparable to the OSI model, the TCP / IP model is another layered communication model composed of four layers: Application, Transport, Internet, and Network Access. These two traditional network models, despite their wide acceptance and general usage in the world, have some problems and have become more and more complex to administer and secure [65]. For instance, if any policy or configuration of the network changes, the network administrators should spend a lot of time manually configuring network equipment through command lines.

In addition, this network equipment is vertically integrated and most often operates with proprietary software [66]. For example, in traditional network equipment, the part which defines and controls the routing of the packets (Control Functionality) and the part which forwards the packets through the network (Data/Forwarding Functionality) is composed of a single device, which makes it difficult to develop. This coupling makes any innovation difficult since deploying any new protocol or service on the equipment network must go through the manufacturer and can sometimes be very long. For compensation for these architectural rigidity problems and many more, SDN was introduced in 2009.

3.2 SDN History

In 2006, SDN was born out of the Clean Slate project funded by the GENI project at Stanford University in the United States. The Clean Slate project's ultimate objective is to redesign the Internet's architecture, with a focus on updating the present network's infrastructure since it has become rather obsolete and challenging to advance. In 2007, a Stanford University student, Martin Casado, directed a project called Ethane which was about network management and security; this project endeavored to practice a centralized controller to let network administrators simply state security control rules that are based on network traffic flows. This is all that is required to achieve security control over the whole of network communication. These security policies are then applied to the different network devices. In 2008, based on the motivation of the Ethane project and its predecessor project called Sane, professor Nick McKeown and his colleague went one step forward and defined the concept of the OpenFlow protocol, and published a paper entitled "OpenFlow: Enabling Innovation in Campus Networks" in ACM SIGCOMM the same year. In this paper, they not only introduced

the concept of OpenFlow, in addition to clarifying the OpenFlow operation workflow, but also descriptions of the many possible OpenFlow application scenarios were also supplied.

Further, based on the programmable characteristics that OpenFlow gives the network, professor Nick McKeown and his colleagues introduced the idea of SDN (Software Defined Network). In 2009, the SDN concept was selected as one of the top ten innovative technologies in the Technology Review of the year. Since then, it has been widely considered and strongly supported by academia and the industry. In December 2009, the OpenFlow specification has become more mature by releasing version 1.0 which can be used in commercial products. For example, it provided OpenFlow's support plug-ins on Wireshark packet capture analysis tools, OpenFlow virtual computer simulation (OpenFlowVMS), OpenFlow debugging tools (liboftrace), etc. At present, the OpenFlow specification and versioning have gone through 1.3, and 1.4, and the latest released version is 1.5.1. The OpenFlow 1.6 standard has been released internally by ONF but it's only available for ONF members.

In March 2011, the Open Network Foundation (ONF) was established under the supervision of Professor Nick McKeown and others. The major responsibility of this non-profit consortium is mainly dedicated to supporting the standardization and development of SDN architecture and technology. ONF currently has more than 96 members, of which 7 are the core members who created the organization, such as Google, Facebook, NTT, Verizon, Deutsche Telekom, Microsoft, and Yahoo. In December 2011, the first Open Networking Summit was held in Beijing. The summit invited all leading local and international companies in SDN to present their successful use cases in SDN; and in parallel, the world's top Internet communication network and The IT company discussed how to deploy SDN-based hardware and software in global data centers, and this was a suitable introduction, also advertising of OpenFlow and SDN concepts in academia and industry.

In April 2012, ONF released the SDN white paper (The title was: Software-Defined Networking: The New Norm for Networks), then the three-layer architecture model of SDN was widely accepted by the industry. In April 2012, Google revealed that its backbone infrastructure network had been entirely functioning using OpenFlow and through 10G network links implemented in 12 data centers around the world. This proved that OpenFlow wasn't merely a research model anymore that stays in the academic scope but already has sufficient technological maturity that can be applied in a production environment.

At the end of 2012, a number of prominent companies such as AT&T, British Telecom (BT), Telefónica, Verizon, Telecom Italia, Deutsche Telekom, and Orange jointly initiated the establishment of the Network Functions Virtualization Industry Association (NFV), initial targeting was to present the SDN paradigm to telecom Industry. Currently, it is formed by more than 52 network operators, telecommunications/IT equipment providers, and technology suppliers. In April 2013, the LINUX Foundation joined the race of developing SDN controllers, southbound/northbound APIs, and other software to break the monopoly of major manufacturers of network hardware, drive network technology innovation and make network management easier and cheaper.

3.3 The Concept of SDN

The SDN was born out of a large intellectual movement motivated by the question of why network equipment should not be programmable like other platforms of computing, and the need to solve the problems that were earlier mentioned in traditional networks, difficulty in managing and developing networks.

SDN is an emerging idea in network management. It's a novel methodology for designing, building, operating, and securing networks. It is based on a logically centralized management network by decoupling the control plane from the data plane, and it makes the network flexible and programmable. The control plane is in charge of defining and associating a routing decision with the data plane packets, and the data plane represents the physical or virtual infrastructure. It is only responsible to take care of the routing and forwarding of the packets into the network. In the SDN structure, the network intelligence is outsourced from the network devices and managed by external equipment called "controller" which manages and controls the roles of the control plane. One or more distributed virtual computers or physical machines may make up this intelligent unit.

All communication between the controller and network equipment should pass through the Southbound API (for example, OpenFlow, LIST, BGP, ...) via a secured channel. The SDN applications with various functionalities, such as policy implementation and management, security services, and network configuration, will mount to the controller via a well-defined Application Programming Interface (API), which is called Northbound API (for example, REST API). In other words, the main objective of SDN is to enable the networks to be agile, programmable, and flexible in order to make their control and management simple [67]. The interest of the big companies that have the main role in the digital world, such as Google and Microsoft [68] [69] in the deployment of SDN concepts into their data centers, gives good prospects for the concept of SDN to be accepted in the real world.

3.3.1 General definitions of SDN

Considering the functionality of the traditional network device, it can be divided into two planes. The data plane and control plane or management plane. The first plane which is called the data plane also can be called the forwarding plane, in this step is where the network data (frames, packets, datagrams) is effectively transmitted through the network by the network device from one node to another. The second plane is the control, which essentially defines and represents the protocols that are used in the network and is able to define the rules for the routing or forwarding table that are ultimately used by the data plane. Finally, the third plane, the application plane; is where any software-based service that is used is included to remotely manage or configure the functionality of the control plane [70].

Through the SDN Architecture, the following four basic characteristics can be achieved:

1) The control and data planes are separated.

The control functionality is removed from the network devices therefore the network devices would only be responsible for the packet, and datagram forwarding (data plane).

2) Forwarding decisions are based on incoming flows and not on destination addresses. In a general perspective, a collection of values within some of the fields which belong specifically to the header of the packets (frames or datagrams) and by a set of actions or instructions respectively defined the flows. These values in the fields of the header consider as the options which make up the filter criteria for making forwarding decisions.

3) The control logic has been delegated to an external unit.

This entity is called an SDN controller, which operates as the Network Operating System (NOS). The SDN controller is a software platform that is running on a server (physical or virtual). It delivers the critical resources, also abstraction to facilitate programming or configuration for forwarding network equipment (such as switches or routers).

4) The application software that operates on the server.

The network Operating system (NOS) is what essentially turns the conventional network into a programmable network. It can interact directly with the network devices of the data plane.

These features offer various benefits to the operational, design, and scalability of the network infrastructure. More specifically, using logically centralized control offers a simplification way for defining and modifying the network policies, which makes it even less prone to mistakes. Additionally, a control program can react to any low-level network changes just by modifying the high-level policies. Furthermore, based on the logical centralization of the control, it contains global information of the whole network, which conclusively can contribute to developing the service functions and more sophisticated network applications.

3.3.2 The major characteristic of the SDN Network

An SDN network architecture has 4 fundamental features:

1. The control and data planes are decoupled. The control functionality is separated from the hardware of network devices, so in this architecture, the network elements play a simpler role which is just transporting data on the network.

2. The network can be programmed which means maintaining, controlling, and programming the data plane will be done by software applications operating above the network controllers.

3. The control logic is moved to a logically centralized entity or network operating system (NOS). A NOS is a software platform that runs on a general server that offers crucial resources and abstractions of the layers to facilitate the programming of SDN infrastructure devices.

4. Traffic forwarding decisions are based on flows. contrariwise to the traditional network, in which all decision policies are based on the network packets, in the context of the SDN architecture, the forwarding decisions are based on flows.

A flow is a series of packets transmitted between a source and a destination that share common characteristics and receive the same policy handling on the network devices. The abstraction of flows makes it possible to standardize the operation of a variety of network devices, including switches, routers, firewalls, traffic optimizers, and load balancers.

3.4 SDN Architecture

As illustrated in Figure 3.1, the architecture of the SDN network is divided into three layers which are called the infrastructure/data-plane layer, the control-plane layer, and the application-plane layer. Communication between these different layers is made through the Southbound, Northbound, and East-Westbound APIs. [71].

In this section, these layers will be explained in detail to provide a better view of the SDN architecture.

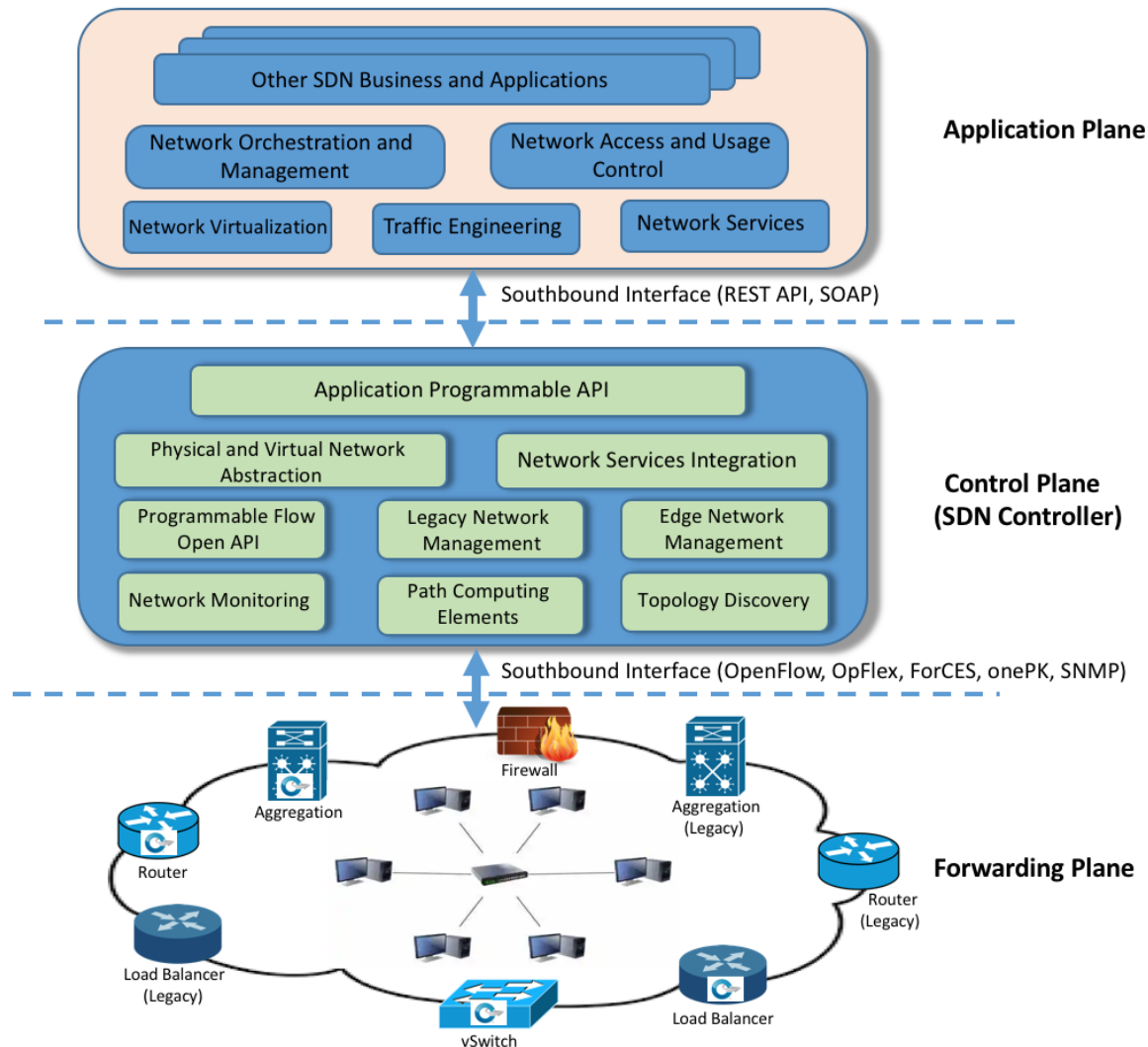


Figure 3. 1: The Architect of the SDN Network

3.4.1 The Application Layer

The application layer is the uppermost layer of the SDN structure where all the management policies are defined and applied. These rules are defined by using a set of applications that are intended to specify an explicit service to the network such as firewalls, load balancing, IDS, etc. This layer makes it possible to define, interact and deploy the behavior of network devices through the SDN controller. As mentioned earlier, in the SDN concept, the control plane

provides a global view of the network and all the requested information about network elements by the applications should pass through the controller's southbound API. Since, the OpenFlow is the de facto standard API southbound in the SDN environment, for defining and implementing network functions and tasks in the form of OpenFlow-based applications, with help of this layer, several services can be implemented above the OpenFlow-based controller as applications.

3.4.2 The Control Layer

In the SDN architecture, the control layer entails a software controller called the SDN controller, which is used to centralize the intelligence part of the network. It is the most important component in the SDN concept, and it can be considered as an equivalent to the network operating system, which is responsible for handling the network management and configuration by defining and installing packet routing rules of the infrastructure layer through the Southbound API in the appropriate network devices (such as routers, and switches). Some of the core functionalities of the controllers are:

- 1- Topology and network management
- 2- Device discovery
- 3- Path computation

Moreover, the controller maintains a global view of the network through its topology discovery service, and this capability is very essential for the accurate operation of the other network services of the controller, such as network configuration. Some controllers are centralized (such as Beacon, Floodlight, POX), and some of them are distributed (such as ONOS, OpenDaylight); therefore, in distributed controllers, there should be an integration of additional APIs for Eastbound and Westbound traffic. Distributed controllers make it possible to overcome the limitation of the use of a single controller, which can not only become a single point of failure from the security perspective but also may cause a problem for the management of vast networks with an enormous number of network entities.

3.4.2.1 The Controller Characteristics

The core functionalities of the SDN controller can be defined as the following mechanisms:

- 1- Topology management, responsible for discovering the topology of the network,
- 2- Device management, in charge of configuring and managing the network infrastructure equipment,
- 3- Path computation, the major task is selecting the best paths (Shortest path) toward destinations,
- 4- Notification management, responsible for managing the communication of the control plane with network elements,
- 5- Statistical information management, responsible for collecting flow traffic measurements from the network,
- 6- Security mechanisms are in charge of providing network protection mechanisms.

3.4.2.2 Different Types of the SDN Controllers

There are currently two types of SDN controllers: (1) centralized SDN controllers and (2) distributed SDN controllers. In a centralized SDN controller design, a single control entity manages all SDN devices. The major limitation behind this type of design is it may cause the condition of a single point of breakdown, which means that when a failure occurs in the controller or in any of its links, the network continues to operate in a restricted way, which makes it impossible to accept any policy changes and new services and flows.

In the distributed SDN controller design, the control of the network is based on several distributed controllers, which offers an enhanced availability of the network in case of failures and provides the network scalability by incorporating additional controllers as required. An outline of Distributed SDN controller can be a cluster of centralized nodes in the same place or a group of geographically distributed controllers. The first option can afford the high performance, particularly in extremely data-dense centers, and the second option can offer higher resilience in the conditions of physical or logical failure. As the third option, In the Cloud communication situation, the Service Providers consist of multiple data centers interconnected with each other through the WAN connection links. In this case, a hybrid SDN controller design can be considered with the clusters of controllers in each data center and distributed controllers in different localities.

3.4.3 The Infrastructure/Data-plane Layer

This layer expresses the functionality of the data plane. The data plane entails network entities such as switches, routers, etc. . A network device is an entity that accepts network traffic(packets) at its ports and executes one or more network functions (for example, forwarding, deleting, or header changing for a specific action) [72].

In general, a switch is a network device whose main role is to transfer the packet. It is made up of two functional parts: the data plane and the control plane. The data plane is in charge for routing and forwarding the packets from the source to the destination. It retrieves the packets from the input interface, then for forwarding the packets, checks its routing table to determine the egress interface. The control plane is in charge for defining the forwarding rules, and building and maintaining the routing table.

The data plane contains of network devices such as switches and routers. The SDN-based network devices are designed to use an open and standard programming interface, which allows them to ensure their configuration and interoperability with the control plane and other SDN switches. There are several standard protocols that can be used such as OpenFlow, LIST, BGP, etc.

There are two types of OpenFlow switches [73]: the OpenFlow switch Only which is specifically designed for SDN networks and for supporting the OpenFlow and the OpenFlow enabled switch which Simultaneously can undertake the role of traditional and OpenFlow-based switches. For this research, the OpenFlow is considered the standard interface between the control plane and the data plane. The OpenFlow-based switch and the controller communicate via the OpenFlow protocol which will be explained in detail subsequently.

3.4.4 SDN Programming Interfaces

SDN has introduced several types of interfaces to allow different elements of its SDN architecture to interact with each other [74]. This section describes the details of these interfaces which are the Southbound API, the Northbound API, and the East-Westbound API.

3.4.4.1 The Northbound API

The communication between the control layer and the application layer is provided through the interface which is called the Northbound API. It is one of the key elements for providing the abstractions between the layers in the SDN environment. It provides the way in which the information can be exchanged between the controller and the applications running on the network. In other words, the Northbound API is an API used by the network administrator to define policies to schedule the control plane. Generally, it indicates a list of basic network functions associated with the provider, which are then used to configure the infrastructure network equipment, and the controller in between is responsible to interpret it into a language that any network device can understand.

The Northbound API represents the network abstraction data model and also controller functionality which can be used by network applications. Northbound API is used to facilitate innovation and efficiently orchestrate the network. The essential of collecting the control information for the network applications from the underlying infrastructure network is another major rationale for this interface. Currently, various types of SDN controllers offer a wide variety of Northbound APIs. The Northbound API mainly can be classified into three categories, REST APIs (Representational State Transfer) [75], specialized ad hoc APIs, and programming languages such as Frenetic[f] [76], Procera[e] [77], FML [78] (Flow-Based Management Language), NetCore [79] and Pyretic [80].

Even though, the ONF organization created a working group 22 to define and develop a standard protocol for this interface, at the time of writing this thesis, still there is no industrial-recognized standard for Northbound API. The REST API is the most used solution as the Northbound API because it provides simple integration and minimum interaction between a client and a server. Obviously, it is not a protocol but an architectural concept intending to facilitate the programming of service-oriented applications by using the HTTP protocol. Moreover, because of this ease of use, many large companies such as Facebook and Google use it to offer their services. This is the main reason that currently most of the SDN controllers use the REST API as a Northbound interface to provide network information to applications [81]. Apart from ONOS and OpenDaylight controllers, most of the other SDN controllers still do not support and entirely implement the REST API, they are still using other legacy methods such as SOAP. this is also one of the main reasons for choosing the ONOS controller for this research.

3.4.4.2 The East-Westbound API

The East-Westbound API is the interface that allows multiple controllers to share a common view of the network and coordinate with each other for implementing the policies and protocols (see Figure 3.2). Through this interface, the SDN controllers can manage the way to interact with each other to share information and consequently, transmit status data about their networks and have an impact on routing decisions. This East-Westbound interface can be used to improve communication across different domains of an SDN network (intra-domain and inter-domain) and accordingly improve the scalability and interoperability of the SDN environment. The East-Westbound API is very important, specifically for large SDN-based networks, which are divided into multiple subdomains, and each subdomain is controlled by an SDN controller [82].

In overall conception, a large network, for instance, is divided into multiple subnets, then to deploy SDN architecture, each subnet should have its own controller, and each subnet has a global view of its sub-network. Therefore, to build a global view of the entire network, the controllers must exchange their network information with each other, including the topology, accessibility, the network protocols used, network status, etc., through the East-Westbound interface. An East-Westbound interface is also important for automating network decisions to reduce the intervention of the operations of the network administrators on large-scale networks. Some of the functionalities of these interfaces are: (1) provide algorithms for data consistency models, (2) provide monitoring and notification capabilities, and (3) import or export data between controllers.

Currently, most of these types of interfaces are available under the license of open-source, which is a huge advantage in the deployment of SDN in large-scale networks. ALTO, SDNi [83], and HyperFlow are examples of East- and Westbound protocols that can be used for compatibility and interoperability between different controllers in large-scale SDN-based networks when using multiple controllers. By using these protocols common requirements can be defined to manage the establishment of the flows and the exchange of reachability information across multiple domains, which provide the ability to create scalable and reliable SDN control platforms.

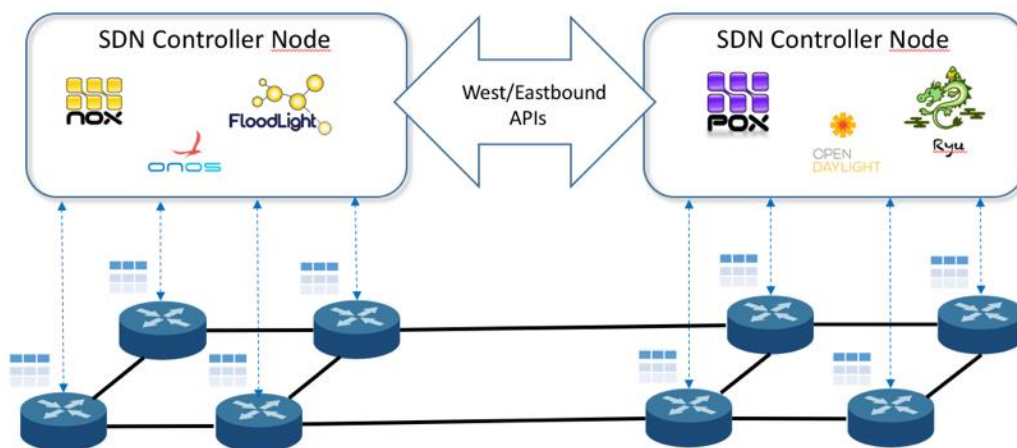


Figure 3. 2: Communication between Controllers via East / West APIs [84]

3.4.4.3 The Southbound API

The interface between the infrastructure layer (Data-plane layer) and the control layer is called Southbound API. The Southbound APIs constitute the communication protocol that facilitates the communication between controllers and SDN network devices. These APIs can be open-source or proprietary [85]. As stated earlier, OpenFlow is the de facto industry standard Southbound API in the SDN architecture which provides communication between the infrastructure network and the controller in the SDN environment. Through this interface, the controller can control and manage all the flows of the switches or routers under its authority. The open flow characteristic and specification will be explained in detail in the next section [86].

Although OpenFlow is the most famous Southbound API and even it is considered the industry standard for controlling and managing the overall information about the network equipment, there are also some traditional protocols that can be considered as southbound protocols in SDN environments such as SNMP [87], and NETCONF [88] (Network Configuration Protocol), LISP(Location Identifier Separation Protocol), and even BGP [89](Border Gateway Protocol). There are also several other specific Southbound APIs to manage communications between these two layers of the SDN architecture, for instance, OVSDB [90] (The Open vSwitch Database Management Protocol), OpFlex [91], and ForCES [92](Forwarding and Control Element Separation).

3.5 OpenFlow protocol

ONF is a non-profit organization consortium of various industrial/Academy members, formed in 2011 that is leading the development and standardization of the critical elements of the SDN architecture, such as the OpenFlow [93] protocol, which provides the standard Southbound communication interface between the controller (control plane) and infrastructure network devices (data plane). The major goal of ONF was to propose specifications to facilitate the configuration of networks, and to simplify management and control networks to deploy the different types of services such as security, QoS, mobile, etc. more easily. OpenFlow is a proper solution for innovating the networks, but it also has to face various challenges, in particular security. The first version of OpenFlow was introduced in 2009 and Figure 3.3 demonstrates the timeline of different versions of it.

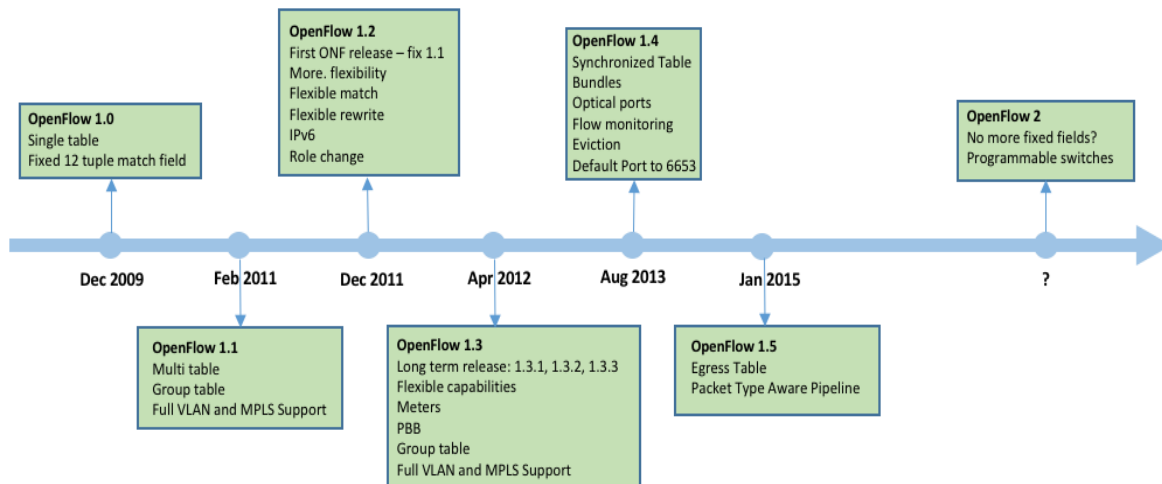


Figure 3. 3: OpenFlow Versions Timeline [94].

OpenFlow [93] is the first interface proposed particularly for SDN, offering granular traffic and high-performance control on multi-vendor devices. An SDN network based on OpenFlow can be implemented using hardware devices (such as switches) and software, offering the following benefits to companies and service providers:

- Centralized management and control of multi-vendor network devices.
- Provide enhancement in automation and management through open APIs, which abstract the details of the underlying physical network, the applications and provisioning systems, and the orchestration of the network elements.
- Speed up the innovation by allowing to offer of new capabilities and network services without having to configure the network devices individually or wait for the release of updates by manufacturers.
- Increase the security and reliability of the network as an outcome of the centralized and automated management part of network devices.

The most advantage of using OpenFlow is to centrally control infrastructure elements, and traffic flows in the network and enables straight access and control of the data plane of underlying network devices, both physical and virtual. In an SDN network, the control plane is decoupled from the physical network and located in a centralized controller. The controller uses OpenFlow to communicate with all components of the network, as indicated in figure 3.4. Through this protocol, the network administrators can manage the whole network rather than configure each device individually.

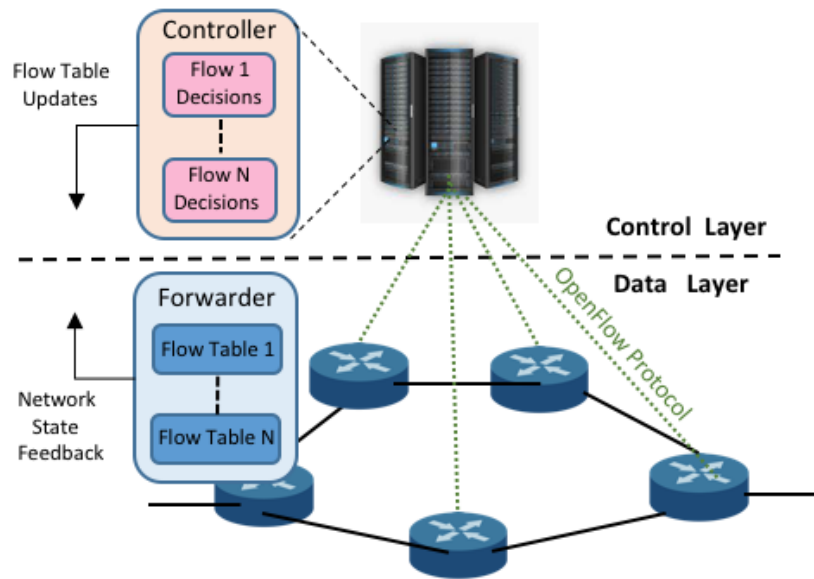


Figure 3. 4: OpenFlow Network Architecture [95].

The OpenFlow protocol should be implemented between both communication sides, the network devices, and an SDN controller. The SDN architecture uses the concept of flows to identify network traffic and makes decisions based on predefined static rules or rules dynamically programmed by SDN control software. Since OpenFlow enables the network to be programmed based on a flow, an OpenFlow-based SDN environment offers granular control of network traffic, allowing the network to respond to changes in real time at the level of applications, users, or sessions.

Table 3.1 expresses the specification of different versions of OpenFlow. The most recent version at the time of this writing is version 1.5.1 (the latest version is 1.6, but it is available only for ONF members).

Table 3. 1: Functionality of the different OpenFlow specifications [96].

Version	Major Feature	Reason	Use Cases
1.0 - 1.1	Multiple table	Avoid flow entry explosion	
	Group Table	Enable Applying action sets to group of flows	Load balancing, Failover, Link Aggregation
	Full VLAN and MPLS Support		
1.1 - 1.2	OXM Match	Extend matching flexibility	
	Multiple Controller	HA/Load balancing/Scalability	Controller Failover, Controller Load Balancing
1.2 - 1.3	Meter table	Add QoS and DiffServ capability	
	Table miss entry	Provide flexibility	
1.3 - 1.4	Synchronized Table	Enhance table scalability	Mac Learning/Forwarding
	Bundle	Enhance switch synchronization	Multiple switch configuration
1.4 - 1.5	Egress Table	Enabling processing to be done in output port	
	Scheduled bundle	Further enhance switch synchronization	

3.5.1 OpenFlow Architecture

The OpenFlow network architecture contains three fundamental concepts: first, the control plane consists of one or more OpenFlow controllers that define, maintain, and distribute forwarding policies into the network infrastructure (data plane), second, the data plane network which consists of the OpenFlow-based devices (switches), and third, a secure control channel which the controller and OpenFlow switches can communicate with each other.

3.5.1.1 OpenFlow Switch

An OpenFlow-based switch [97] is a network device that forwards the incoming packets based on the rules defined in its internal flow table. The flow table in the SDN switch contains a set of flow entries consisting of the header, counter, and action fields. The header field is used to determine the match of incoming packets/flows with any entries of the flow table. The header field can accept different types of network protocols, depending on the version of the OpenFlow, such as Ethernet, IPv4, IPv6, or MPLS. The counter field is used to collect statistical information from flows, such as the number of bytes of received packets/flows or the duration of remaining flows in the tables. The action field defines the treatment which should be done to incoming packets, such as forwarding packets to a port, modifying a packet, or discarding a packet.

3.5.2 OpenFlow Workflow

As mentioned earlier, in an OpenFlow-based SDN architecture, the data of the network (packet/flows) are forwarded through the switches based on the forwarding decisions which are made in network operating software (NOS) is implemented on an external machine(server) called controller that communicates with the switches through the OpenFlow protocol.

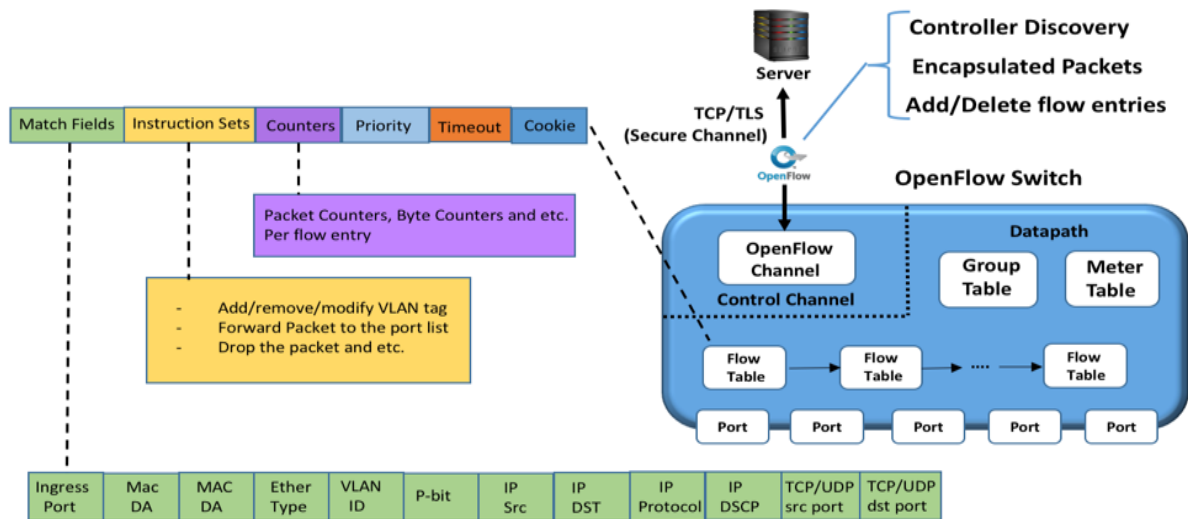


Figure 3. 5: SDN / OpenFlow device [98].

Through OpenFlow protocol, the control and underlying network devices can communicate with each other. It defines the specification and format of the exchanged messages between the SDN controller and the OpenFlow switch and uses TCP protocol for exchanging these messages. The OpenFlow protocol provides the possibility for the controller to modify, add, update and delete input flows in the flow table of the switches. The messages between the OpenFlow controller and the OpenFlow-based switches are exchanged through the secure channel and implemented via an SSL/TLS connection over TCP. When the switch distinguishes the controller's IP address, it initiates the SSL connection.

A controller may create, update, and remove flow entries from the network switches' flow tables using the OpenFlow protocol, proactively or reactively in response to the arrival of packets, modifying the forwarding action of the data plane of the switches (Figure 3.5). When a network packet reaches the one of switch ports, the switch starts a process of querying the flow table to find out if there is a flow entry in the table that matches the packet. The flow entries are evaluated in priority order, and the first match in the table will be considered. If there is a match in the table for the arrival packet, the actions indicated in the flow entry would be executed for that packet, and the counter field of the flow entry correspondently is updated. In case there is no match in any entry in the flow table inside the switch, the packet is sent to the controller over the OpenFlow communication channel (Figure 3.6).

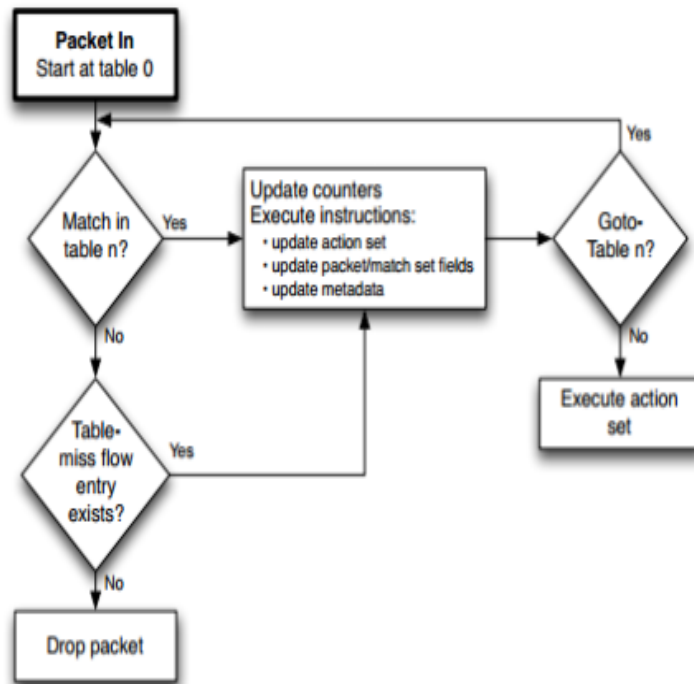


Figure 3. 6: OpenFlow Switch Operation workflow [99].

All messages exchanged between the controller and the switch begin with an OpenFlow header consisting of the version of the OpenFlow protocol, the type of the message, the length of the message, and the message identifier. In the OpenFlow specification, there are three types of messages: Controller to switch messages, Symmetric messages, and Asynchronous messages.

Controller-to-switch messages: These types of messages are utilized to directly manage the OpenFlow switches and are basically initiated by the controller. This type of OpenFlow message, which is exchanged between the controller and the switch, is used to detect the switch functionality, information retrieval, configuration, and programming. These messages are, among other commands such as switch configuration, the commands from the controller, statistics, queue configuration, and barrier.

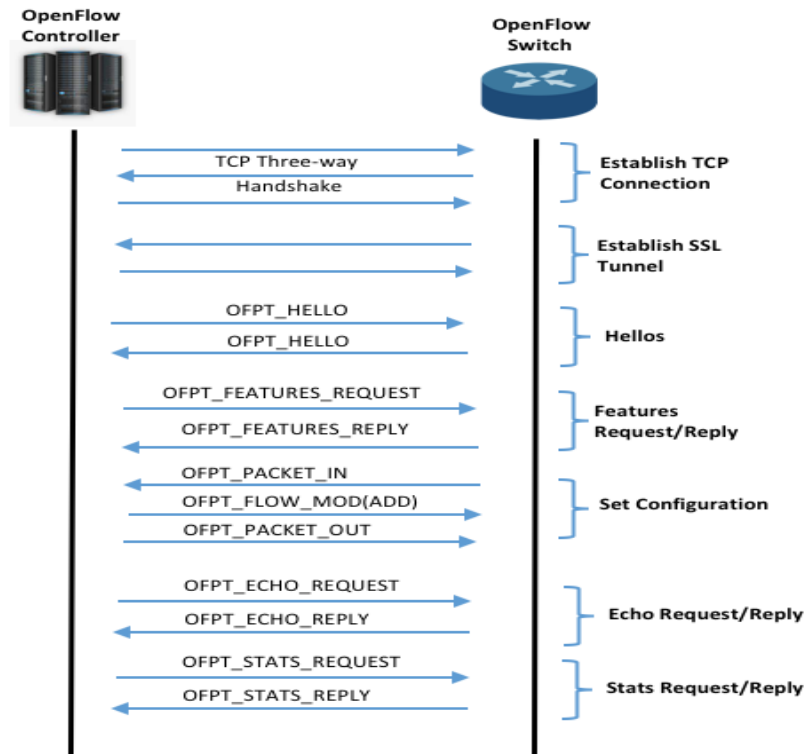


Figure 3. 7: Exchanges the OpenFlow messages between the controller and the switch

Symmetric messages: the HELLO, ECHO REQUEST, ECHO REPLY, and VENDOR are the main types of symmetrical messages which can be exchanged between the controller and the OpenFlow switch. They don't need to be solicited for action to be started by the controller or the switch. After establishing the SSL/TLS secure channel with TCP protocol, the HELLO messages are exchanged between the switch and the controller to determine the version of the OpenFlow protocol which is used. The ECHO messages (ECHO REQUEST and ECHO REPLY) are also used by the switch and the controller while they are operating, monitoring if the established link is still active and, at the same time, checking the speed of the connection and measuring latency. Figure 3.7 indicates different OpenFlow messages exchanged between the Controller and the infrastructure switch.

Asynchronous Messages: Asynchronous messages can be initiated by the switch without any request from the controller. They are used to inform the controller of incoming traffic, switch state changes, and errors. For instance, PACKET-In is the type of message used by the switch to send the flows to the controller for decision-making. This type of message is sent, when none of the entries in the switch's flow table matches the incoming packet, or when it is indicated at the action level of the corresponding entry in the flow table that the packet is to be transferred to the controller. In general, the traffic from the data plane is transmitted to the controller via the PACKET-IN message. Figure 3.8 shows a sample flow sent to the SDN controller via the OpenFlow channel.

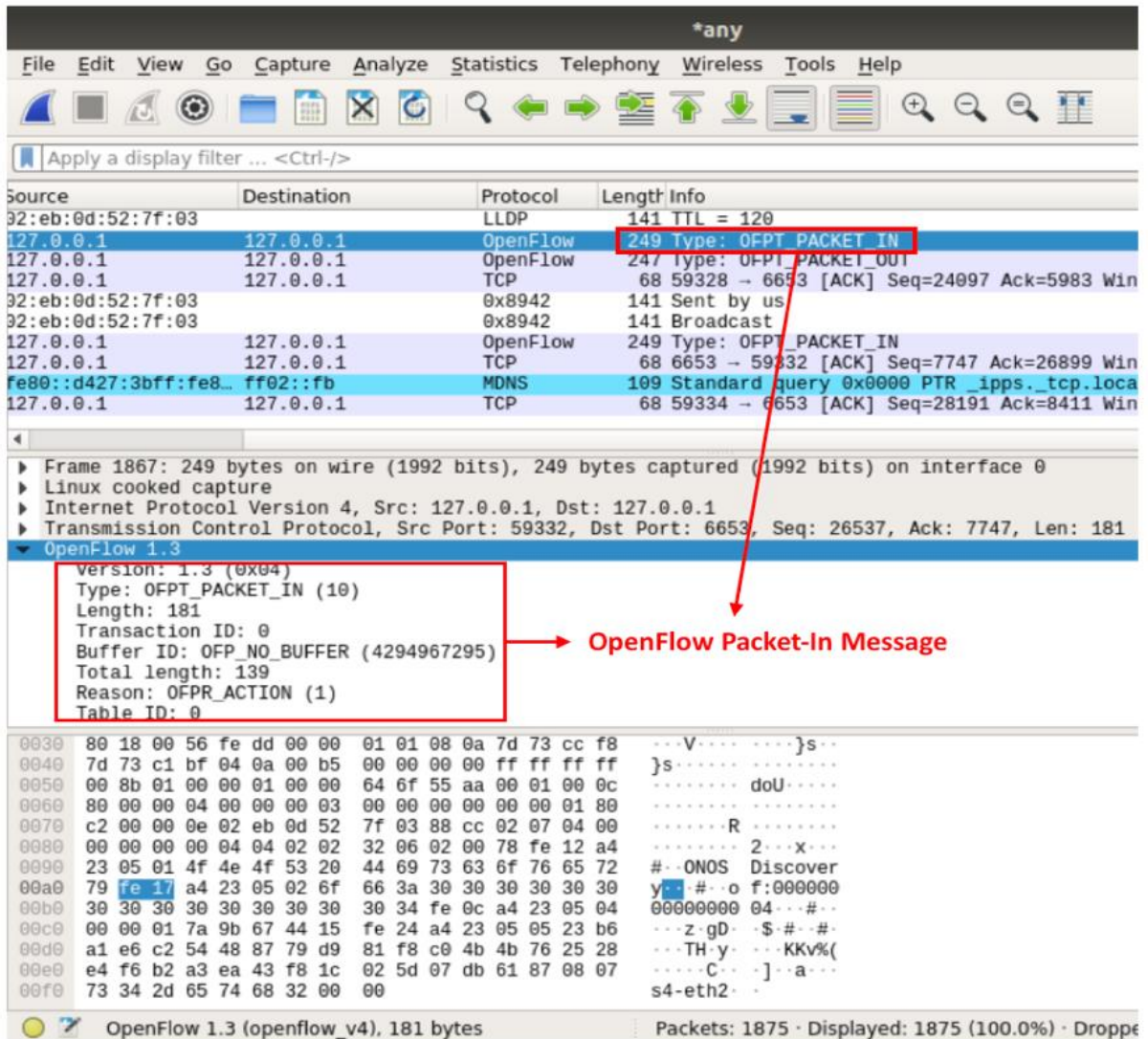


Figure 3. 8: Sample flow sent to the controller via OpenFlow.

On the other hand, the FLOW-REMOVED message allows the switch to inform the controller in the event of a deletion of an input flow from the flow table. The switch removes an input stream when no incoming packet matches this entry for a defined time by the controller when creating this input flow in the switch flow table. PORT STATUS is the message that allows the switch to inform the controller of any changes in the configuration or port status of the switch. The ERROR message is used to alert the controller for occurring the errors. For example, the ERROR message is sent to the controller when the latter tries to add a flow entry containing actions not supported by the switch.

3.5.3 Flow Load Modes

There are two methods to define and specify entries in the flow table of the switch: Reactive Mode and Proactive mode.

Reactive mode: In this case, when the new packet arrives at the switch, it triggers a call to the controller via the OpenFlow channel so that the controller can create and insert a new entry(rule) in the flow table of the switch which refers to the new package.

Proactive Mode: Under this scheme, the controller can populate the flow tables of the switches before the new traffic arrives, therefore in case of losing the communication channel (OpenFlow connection) between the controller and switch, the network performance would not be interrupted.

In the OpenFlow network one of the following actions can be applied to every incoming flow: (1) forward the packet to a port (destination port), (2) modify the packet header fields and (3) remove the packet.

3.6 SDN Security

After introducing the most essential elements of the software-defined network structure, explaining them and their main characteristics, as well as the differences that exist with the traditional networks, this chapter is analyzing the security challenges which expose by implementing the SDN networks. Similar to any other area or network structure, security is the most important aspect that must be considered since it can become future weaknesses of that network which ultimately end up putting the whole organization at risk. With the rise of modern network concepts that progressively support consistent and dynamic applications, such as the Internet of Things, social networks, cloud services, mobile apps, etc., it is necessary to develop technologies that can adjust themselves to the complexity required by upstream applications to provide security.

In the context of the SDN network, various challenges arise in terms of security. Currently, SDN is mostly used in data center design, therefore its security issues must be deeply considered and put more effort to strengthen it. For this reason, there is a security group arranged by the Open Networking Foundation (ONF), which is responsible to find specific security issues that require further research. There are several types of vulnerabilities that have been addressed in each layer of the SDN architectures. One of them is the necessity of authentication and authorization between application and controller to use the network resources. The problem lies in providing different organizations required to have access to the network resources in a secure way, but not all applications have identical privileges to utilize the network resources; therefore, attackers can perform spoofing attacks to exploit the identity and gain unauthorized access to the resources.

To understand security in SDN, first, the security properties which are associated with a computer system should be declared. There are three essential security attributes that must be provided for each computer system to make it safe: confidentiality, integrity, and availability. In addition to these features, other attributes such as authenticity, accountability, and non-repudiation are also usually included. Confidentiality means that the information must be accessible only to those who are authorized. It guarantees that private or sensitive information is not displayed or accessible to unauthorized users. Integrity provides the opportunity for the information and system function to remain unaffected by malicious attempts. Availability ensures that the computer system can continue to work without

suffering any data or system access degradation and, in addition, offer authorized users the resources which they require whenever they need them. Authenticity ensures that users can be verified as they claim to be and the system receives the traffic packets from trustworthy resources [100].

However, the specific characteristics and the architecture of the SDN itself introduce new security challenges and additional surfaces for attacks that do not exist in conventional networks, aside from security concerns that have already existed in traditional networks. Therefore, it can be seen that in the SDN environment, security has a double connotation: first, the utilization of the SDN characteristics to provide mechanisms for detecting and mitigating methods against known security threats, either by introducing new security offers or by expanding the functionality of the existing defensive systems and security devices. Second, develop the SDN architecture to provide proactive behavior against new attacks and security threats that are raised by the SDN architecture itself [101].

3.6.1 SDN Challenges and Issues

The intention behind the SDN advent is to change the way that networks are managed. The implementation of this network architecture would reduce complexity and increase the efficiency of administrative processes and optimize network management, especially in the large network infrastructures owned by service providers. In addition, it would help to reduce significantly administrative costs in enterprise business and service provider networks. On the other hand, by implementing SDN architecture and separating the network plane from the data plane various new challenges may arise.

3.6.1.1 Scalability

The presence of a centralized controller in the SDN network makes it necessary that when a packet receives into the infrastructure devices (like switches), the network device looks into its flow table for an entry that corresponds to the received package and takes action associated with this entry. If no entry is found in the flow table, the packet is considered as a new packet and will be sent to the controller, and the controller must decide and define a rule based on its policy that which action should be considered for that packet. Therefore, in the SDN architecture, the controller and devices exchange not only the network traffic but also control messages and network information would be transmitted. On a small-scale network, this mechanism performs well, but in a large-scale network, infrastructures may cause latency problems because of the large number of nodes that can exist in the network. This can be a challenge that rises with implementing the SDN architectures in a large network.

Various approaches have been proposed to overcome this particular challenge. One of them, for instance, proposes a strategy based on a group of controllers who are responsible to maintain, controlling, and communicating with the elements of the network infrastructure. As a matter of fact, this scheme introduces a different problem in terms of scalability for the SDN architectures, which means how communication should be established between controllers, and define specific policies for using the east to west traffic (east to westbound APIs). This concern arises due to the principal design of the SDN architecture which each controller requires to have a complete view of the network [102].

The scalability of the SDN network has been considered by researchers from the beginning of the advent of the SDN concept. Various research groups and institutions have been trying over years to find a possible solution to provide scalability for the SDN environment. One of the proposed solutions is developing the HyperFlow framework [103]. HyperFlow is a framework that provides a distributed control plane based on the selective propagation of the events that change the configuration state of the network. Each controller transmits the published events to reconstruct the state of the network to another controller, thus all controllers can share a consistent state of the network.

3.6.1.2 Performance

Another concern about the SDN network is performance management in packet processing. In an SDN network, it is important to efficiently manage the process of the network flows with the necessity of high security and low latency. In this case, two features should be considered to assess the processing performance. One factor is to consider the throughput and associated latency when processing the incoming flows. The other factor is the programmability of the network devices, this feature refers to the ability of the devices to accept the sets of instructions that change the functional behavior of the network. There is another feature that might increase the processing performance of the network devices is the highest transmission speed at which the network connection links can be reached [102].

3.6.2 SDN Vulnerabilities

SDN, similar to any innovative technology, has its pros and cons. Regarding security, as its pros, the SDN technology can be leveraged to fully mitigate some attack threats and vulnerabilities that commonly appear in conventional networks. On the other hand, the SDN technology hosts new risk vectors and vulnerabilities that are inherent in its novel architecture. The decoupling of the data and control planes and the fact that the intelligence of the network is logically centralized makes it vulnerable to being abused as a single point of failure, which might result in the whole SDN network being compromised. As in conventional networks, each network element such as network protocol, network device, or network layers that participates in the SDN network can potentially be the subject of intentional exploitation and this misbehavior can cause system and network failures. This explanation affirms that each element or layer which is a part of the SDN architecture can be considered a threat vector or attack surface; Therefore, any misconfiguration or improper deployment of any elements of the SDN network has the potential to introduce new vulnerabilities and jeopardy of security [104].

The major security problem of the SDN architecture is by default there is no security anticipation in its design. As SDN technology use cases progressively increase in the IT world, the list of security challenges that need to be considered and then prevented is expected to grow. Security vulnerabilities in SDNs are concentrated in the three planes or layers (application, control, and data). Therefore, in the SDN environment, all three architectural layers (Data-plane, Control-plane, and Application-plane), and interfaces (northbound and southbound) are susceptible to different types of attacks that can compromise the network components that reside in the layer or target the elements of the other layers [105].

in this research, for investigating the security issues in the SDN environment, the study of attack vectors is separated into 3 parts: The data plane, control plane, and application layer, and the following paragraphs explain different security vulnerabilities which could occur in each of these layers.

3.6.2.1 Attack on the Data-Plane Layer

In this layer, attackers can target network devices directly from inside of the network. An attacker could theoretically have unauthorized access (physically or virtually) to the SDN network or compromise a host that has been already connected to the network and then try to perform malicious behavior to threaten network infrastructure or elements. This behavior can be a type of denial of service (DoS) or a fuzzing attack that attempt to attack network components. As mentioned before the Southbound *APIs* and protocols are used by the controller (control plane) to communicate with the infrastructure layer and network equipment (data plane). There are various southbound protocols such as OpenFlow (OF), OVSDB, PCEP.

Each of these protocols has its own methods to provide a secure communication channel between the control plane and the data plane. Currently, SDN architecture is mostly used in designing modern data centers, and for providing connection between two or more data centers the Data Center Interconnect (DCI) technology is used. In this technology, besides of underlay network, network overlay is also implemented using various protocols such as NVGRE, STT, VXLAN. These new protocols could be vulnerable due to their natural design or the way the provider or customer has employed them. Therefore, in this layer, an attacker could be motivated for impersonating another network device to modify the links DCI or generate a DoS (or DDoS) attack on DCI connections [106].

3.6.2.2 Attack on the Control Layer

By considering the definition and role of the controller in the SDN architecture, obviously, it can be an important target for the attacker for different purposes. If an attacker can take control of the SDN controller, he would have the ability to instantiate new flows by forging information that is sent from applications via the northbound interfaces to the controller or spoofing packets that the controller sends to the data plane devices through southbound API.

In addition, an attacker may force the controller to reject requests for some services or even attempt to use specific methods to bring down the SDN controller which means that it may cause to bring down the whole network. Furthermore, attacking the SDN controller may cause the controller to consume many resources of the computer system (such as CPU and memory), and therefore, it would respond very slowly to incoming requests from the network. The delayed response to any legitimate traffic from the network may cause a big concern for the SDN network [106].

3.6.2.3 Attack on the Application Layer

As an application layer, there are numerous applications that are used by SDN controllers for different purposes such as network management, QoS, and security. Different Northbound

APIs can be used for communication between the application layer and control layer, such as Java, Python, REST, JSON, and XML. As these Northbound APIs have their known vulnerabilities, if an attacker can use these known vulnerabilities for a malicious attempt, with complete access to the controller, he might be able to control the entire network. According to research, the main threats in the plans or layers of SDN can be categorized and presented in Table 3.2 [106].

Table 3. 2: Main Threats in the SDN Layers [106].

SDN Layers	Type of threat	Description
Application Plane	Lack of authentication and authorization	There are no robust authentication and authorization mechanisms convincing for applications.
	Insertion of fake flow rules	Malicious applications can generate false flow rules.
	Lack of access control	Difficult to implement access control for applications from third-party.
Control Plane	DDoS attacks	The nature of the architecture and centralized intelligence control, and limited resources of the control plane can be the perfect target for DDoS attacks.
	Unauthorized access to controller	Lack of realistic mechanisms to impose access control.
	Scalability and availability	Lack of scalability and availability for the centralized controller causes security challenges.
Data Plane	Fake Flow Rules	The data plane is more vulnerable to fake flow rules.
	flood attacks	flow tables of the switch have limited capacity for storing flow rules.
	Controller Hijack	The data plane operation depends on the security of the controller.
	TCP-Level Attacks	TLS (Transport Layer Security) is susceptible to TCP level attacks.
	Man in the middle attack	Due to the optional use and complexity of TLS.

3.6.3 SDN Attack Vectors

In the world of the network, especially the SDN network, introducing new architecture and defining new components such as interfaces, applications, and controllers faces new security challenges. Compare to the traditional network, security in an SDN network is more complex considering these new components, the central controller can be an appropriate target for performing attacks, also using some open source applications and interfaces makes it difficult to define security policies. Since the SDN controller is the fundamental component of the SDN network architecture, launching an attack targeting the controller can affect the operation and performance of the entire network [107].

Different types of known attacks can be threatening the SDN network such as denial of service (DoS) to disrupt the availability of network services for legitimate users, the man-in-the-middle attack (MTM) to modify the rules sent from the SDN controller to the data plane network devices to take control of the network, and exploiting vulnerabilities of the controller and installing malicious applications to take control of the entire network infrastructure. The SDN architecture was introduced without any security facility in its design. Therefore, mitigating the risk of attack in the SDN network requires a protection-in-design approach to provide a suitable defense against different network attacks and other unintentional security issues such as bugs in installed software, or device misconfiguration [107].

SDN architecture has a characteristic that can turn it into a perfect target for different attacks: the infrastructure network is based on a centralized control entity that can become a target of the attack to take ownership of the system and control of the entire network software that may contain application bugs and other vulnerabilities. Kreutz et al. [116] make deep research focusing on the security vulnerabilities of the SDN architecture. They classify SDN security issues in seven attack vectors (see Figure 3.9). In addition, Hori et al consider the attack vectors indicated by Kreutz et al. [108] and merge them with the list of attacks (using SDNSecurity.org) and make a check sheet to assess the security of the OpenFlow-based SDN network together with its protection countermeasures.

Their study defines seven categories of security issues in the SDN network: unauthorized access, data leakage, data modification, malicious applications, denial of service, configuration problems, and systems-level security. The authors of this research also propose solutions to overcome these problems. As a solution, they suggest the replication of the controllers and management applications to provide alternatives in the situation of any hardware or software failure.

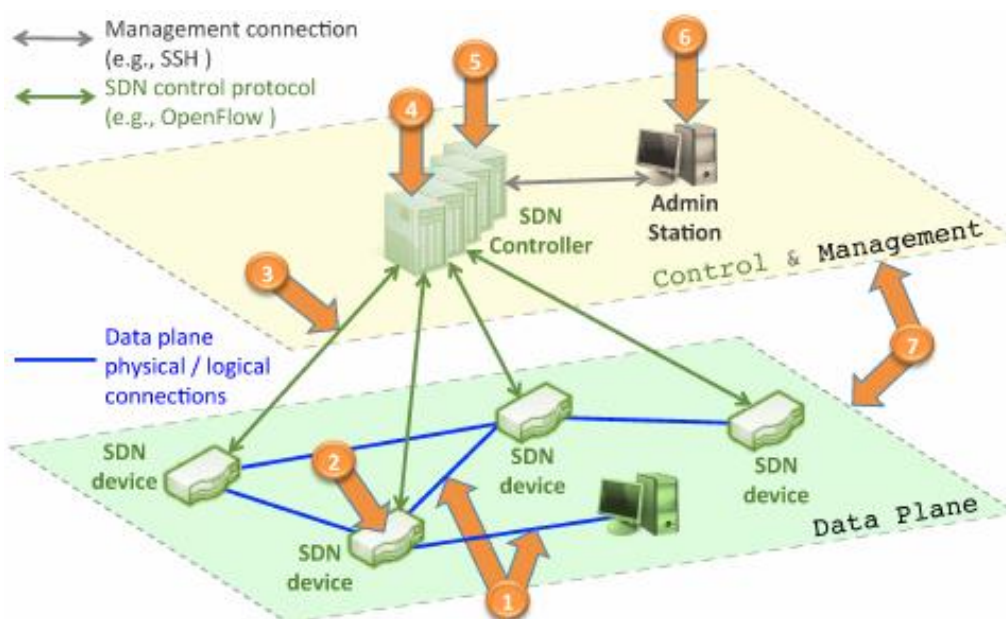


Figure 3. 9: SDN Security Vectors [Taken from [108]].

Threat Vector number 1 which is forged or fake traffic: In this case, an attacker tries to perform an attack by sending fake traffic directly to network devices such as switches. The attacker can use, a workstation, a server, or even another network device such as a switch to launch a denial-of-service attack on OpenFlow-based switches for saturating the southbound bandwidth or controller resources to disrupt the network operation.

Solution: For detecting and mitigating this threat vector, intrusion detection systems can be used to identify anomalies or fake traffic. In addition, this solution can be accompanied by using mechanisms to limit the rate of requests which are sent to the controller to restrict the size of the malicious traffic [109].

Threat Vector 2, Vulnerability attacks on switches:

It consists of using a switch to discard or slow down packets which simply produce a catastrophe in the infrastructure network. A single switch might be exploited to discard network packets, distract network traffic (for example, to facilitate data theft), or even inject traffic or fraudulent requests to overwhelm the controller or nearby switches.

Solution: Implement applications or tolls for testing, including autonomous trust management mechanisms for software components. In addition, using mechanisms to monitor and detect all network traffic to detect malicious behavior in the network can be another solution [109].

Threat Vector 3, Attacks in the communication channel of the plane of control:

The intruders can generate a denial of service attack or data theft targeting the southbound between the control plane and the data plane. In SDN architecture, SSL/TLS technology is used to provide a protected connection channel between the control plane and the data plane. But, SSL/TLS protocol is exposed to the risk of a man-in-the-middle attack. Therefore, the TLS / SSL method is not sufficient to establish a secure channel and ensure trust between the control plane and data plane, more precisely between the controller and switches. Once the intruder gains access to the control plane, the attack level can be extended on several switches under the same domain, and the intruder can easily perform distributed denial of service attacks.

Solution: One conceivable solution can be implementing several trust-anchor certification authorities (for instance, one for each sub-domain or one for each controller instance). Another solution method is to use threshold cryptography to secure communication among controller replicas [110]. Moreover, implementing a trusting relationship between the control plane and data plane devices with the usage of automated, dynamic, and ensured device association techniques might be another feasible solution.

Threat Vector 4, Attacks on the Controller Vulnerabilities:

This is the most severe threat to an SDN network. In the case of a controller failure, it may cause compromise the entire network operation.

Solution: To make the controller secure, various techniques can be used, including controller replication and diversity using backup/recovery techniques to refresh the system into a clean and reliable state in case of any failure, secure sensitive controller objects using cryptographic

methods, implement access restriction policies into interfaces or applications which can manipulate network policies rules, or restrict the scope on the type of rules that they can generate to schedule the network [108].

Threat Vector 5, Lack of mechanisms to guarantee the trust between the controller and applications:

Techniques for verifying devices are different to verify applications, and for communicating between the control plane and application plane, there is no procedure in place to build a trusting connection between the controller and applications.

Solution: Provide independent management mechanisms to guarantee that the application is reliable throughout its life cycle [108].

Threat Vector 6: Vulnerabilities and attacks on the administrative system: The major target, in this case, is to attack the administration consoles of the controller to reprogram network policies from a single point.

Solution: to prevent this threat, a double credential verification protocol can be used, it means that to have access to the management server, the credentials of two users are required. Additionally, in this case, recovery mechanisms are also required to let the management server return to a reliable state after a system failure.

Threat Vector 7: Lack of reliable resources for forensic activities and remediation: To investigate and determine the facts about an incident in the network, trustworthy information from all the elements and resources of the network is required. This information is useful if they are trustable and have been previously authenticated and kept its integrity. Remediation requires secure and reliable system snapshots to ensure fast and correct recovery of the network elements to a known state.

Solution: Implement logging and tracing mechanisms in data plane devices. data and control. In addition, the records must be persistent and must keep a backup of the information in an external storage repository [108].

3.6.4 Improve Security in SDN

As mentioned above, the SDN architecture and separating the control plane from the data plane introduces new vulnerabilities and security. Although the introducing of new interfaces and layers by SDN architecture simplifies the network management and control, it provides the opportunity the new surfaces of attack and exploitable targets. In other words, despite providing the facilities to improve network security by utilizing the new features and novelties concepts of SDN, the network infrastructure architecture could persist insecure if the SDN architecture itself would not completely secure against security threats introduced by SDN features.

The channels and interfaces used for the exchange of information between the three layers in the SDN environment can be a proper target for the attacker if they are not accurately protected, the exchanged data through these interfaces can be eavesdropped on to compromise Network entities. Therefore, in this case, all data which passes through

communication channels and interfaces of the SDN network must be encrypted, so that if an attacker manages to gain the data exchanges, it cannot extract the real information. Moreover, besides the encryption of the interfaces, the security of the SDN network especially at the control plane should be strengthened by using authentication mechanisms. The SDN controller must be able to identify and authorize trusted devices (such as other controllers, and switches), to ensure that only trusted devices and applications can access network resources and malicious devices and applications persist isolated from the network.

It is essential to consider the protocols and services used by the communication channel before selecting the protection mechanisms to make the connection secure. In order to achieve a high level of security for different layers of the SDN architecture (data plane, control plane, and application plans) various mechanisms (for instance, limited access to management services and applications, authorization and authentication methods, security assessment) are required to be developed for detecting and mitigating malicious attempts launched by untrusted users [111].

3.7 Denial of Service Attacks

Denial of Service or DoS attacks has been growing in recent years and has become a real threat and challenge in the area of security in the IT world. It can target various parts of society such as companies, government agencies, banks, armies, or public services such as universities, hospitals, and airports. According to the report [112], currently, in the field of cybercrime, this type of attack is the first place IT threat for companies in the United Kingdom and the second place in the United States.

The most effective and widely used DoS attacks are of a distributed form of it which are known as DDoS (Distributed Denial of Service). In this case, the attacker uses a computer system as a master server (called botmaster) to control several previously infected computers called slaves or zombies. In this type of attack, the attacker creates a network of zombies to attack their victims in a synchronized way. This scheme is shown in figure 3.10.

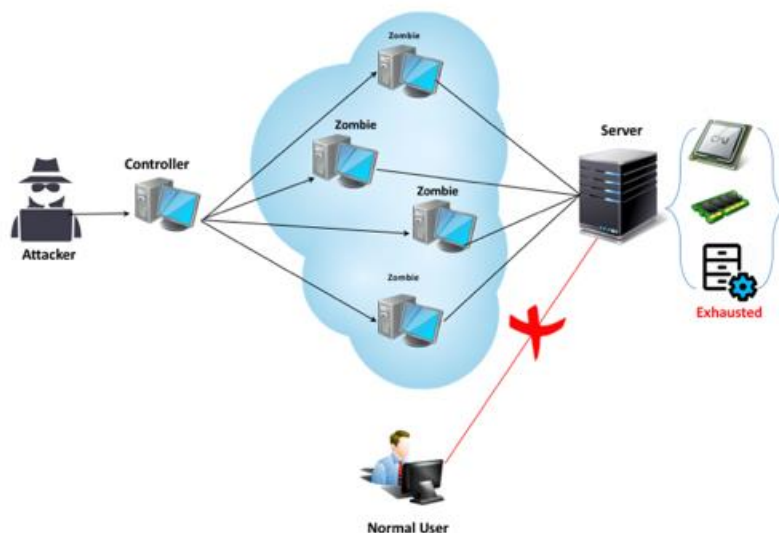


Figure 3. 10: DDoS attack

Denial of service attacks is a type of malicious attempt to cause an interruption or suspension of one or several services. This objective can be achieved through the excessive consumption of one or several resources that the target server or other victims such as network infrastructure devices. The malicious activities of the DoS attack which can break down the resources of the target victim are CPU over-processing, memory overloading, making the disk or database capacity full with fake data, saturating the bandwidth of the connection, and filling the inner tables of the network device with unnecessary information. Various methods of sending massive network traffic are used to achieve the above-mentioned outcomes. This technique is called flooding.

During the flooding attacks, after some time the target or victim cannot respond to the requests because the traffic is too heavy in quantity and disrupts victim resources, another approach to achieve this outcome is to exploit vulnerabilities in target operating systems or their applications. A Distributed Denial of Service (DDoS) attack is a kind of malicious behavior that attempts to disrupt the normal services of a server, to overwhelm the target or its surrounding infrastructure. In this attack, the more computers are used against a particular target for performing the DDoS attack, the more powerful and effective the attack would be.

To maximize the influence of DoS attacks, the attackers introduce a much more effective technique: Distributed Denial of Service or DDoS attacks. DDoS attacks are defined as: “attacks that aim to exhaust the critical resources of a system/network and that come from multiple sources distributed throughout it” [113]. To launch a DDoS attack, a network of various computer systems infected with malware which is called a botnet. Once the botnet has been established, the attacker will attack the target server (using its IP address), and by sending a huge number of requests, try to overwhelm the target. In this type of attack, separating the attack traffic from normal network traffic can be quite problematic, since each bot machine can be a normal and legitimate Internet device. This means sending large amounts of malicious traffic (currently, on average the size of Tbs/Pbs) to the target victim. For intruders, this technique provides more advantages over conventional DoS attacks, including:

Combined attacks: when a DDoS attack is executed, each source (Zombie) generates individual malicious traffic, but as they employ a similar type of DDoS attack (same target, IP address, and protocol), eventually the whole accumulated traffic will be considered as a threat.

Simplify of Hiding: In this attack, the real attacker can easily hide behind the infected computer under his control and it is difficult to find the real intruders. It seems it is just the normal behavior of enormous legitimate users attempting to access the targeted server.

Multiple sources: It is difficult to identify this attack since the source of the attack is distributed between a vast group of computers.

In the traditional network architecture, a connection between a computer system and a server is defined by the OSI model. The DDoS attacks can target various layers of this connection model. Therefore, to understand the different types of attacks, it is first necessary to mention the connectivity layers of the Open Systems Interconnection (OSI) model, ISO/IEC 7498-1 [114]. According to the report presented by the F5 Silverline Security Operations Center (SOC)

[115] The distribution of DDoS attack techniques observed by the SOC team during a 13-months period is shown in figure 3.11. At first look, volumetric DDoS attacks (UDP fragmentation, DNS reflection) seem to be the most prevalent.

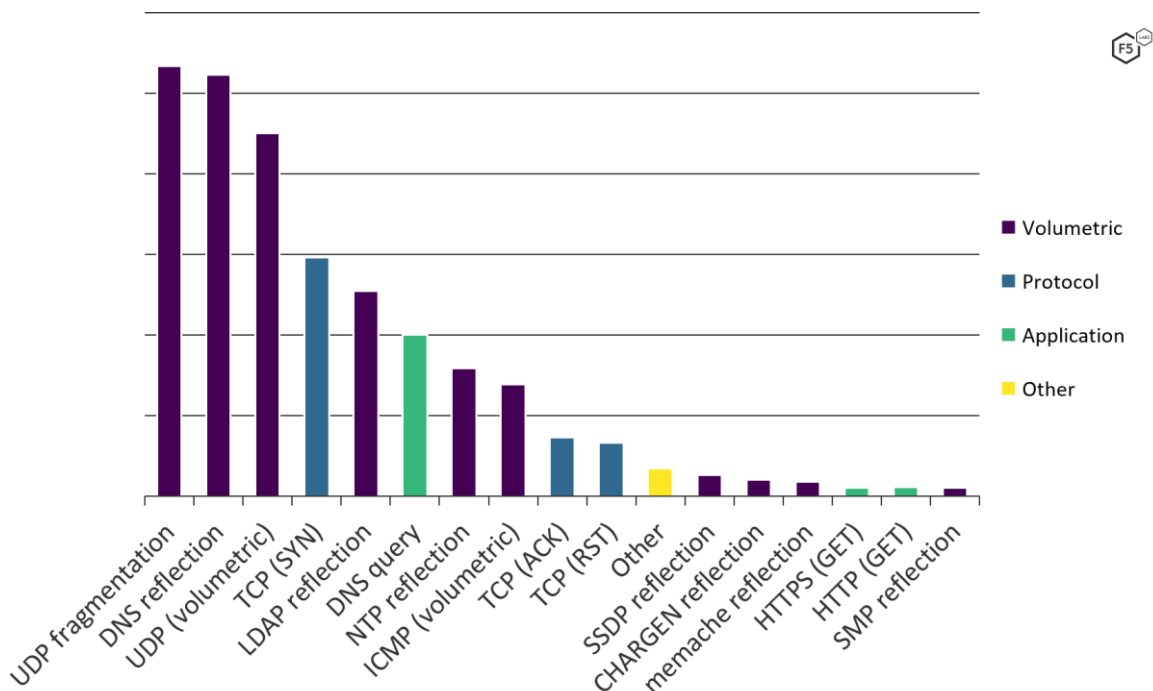


Figure 3. 11: The incidence of various DDoS attacks from January 2020 to March 2021. [Taken from [115]]

The above-mentioned described DDOS attack characteristics indicate that this kind of attack can be a potential threat to any type of network. The spread of the Internet in the world allows various network infrastructures to be connected to each other without implementing proper security policies which makes the whole network to be vulnerable to traditional attacks. On the other hand, there are tools that can be found to offer the possibility for any user to launch this kind of attack without any need to have high technical knowledge.

The major concern about DDoS attacks is this kind of attack is very difficult to be detected because their behavior is in a way that the normal user generates a legitimate request and valid traffic, because of this feature when a DDoS attack occurs, the malicious packets can be considered as legitimate traffic, and not a potential threat. Another concern that should be considered is to detect and prevent the DDoS attack accurately, all the incoming packets to the network should be evaluated, and this can cause overloading of the responsible device’s resources, consequently making it become a bottleneck for the network operation and trigger a long delay for response times to the normal network request.

The advent of the SDN concept and its characteristics propose new features which can be used as an opportunity for detecting and mitigating the DDoS attack. Two major features which can be addressed here are the centralization of the control, and the possibility to have knowledge about the state of the entire network. These features can be used in the following way for detecting DDoS attacks: Utilize the controller for requesting the sample of the incoming flow, evaluate the flow headers, classify the new flows as normal or malicious traffic, and finally

define, produce, and send rules to network devices (such as a switch) to eliminate hazardous flows.

3.7.1 Different Types of DDoS Attacks

DDoS attacks are classified into different categories, each category targets a specific component of the infrastructure network. DDoS attacks can be classified into three types: volume attacks, application-layer attacks, and protocol attacks. An attacker may use one or more of the three techniques to attack different elements of the network.

3.7.1.1 Volumetric Attack:

It is the highest prevalent type of DDoS attack, and since the range of its usage is growing drastically fast, this type of DDoS attack receives the utmost attention from researchers and network security managers. The method which is used to launch this attack is sending a gigantic stream of worthless but simple packets of information to the targeted system, requiring the target to handle an enormous quantity of data and simultaneously using all the available bandwidth between the victim's system and the Internet to cause congestion.

Amplification forms or other techniques of creating enormous traffic, such as requests from a botnet, are used to send huge volumes of data to a destination. A well-known instance of a Volumetric attack is DNS Amplification. In this type of attack, an attacker directs a request with a spoofed IP address to the DNS server, as a result, the victim receives a huge number of responses that it certainly not requested from the server, and will ultimately crash.

3.7.1.2 Application Layer attack:

This kind of malicious behavior is frequently referred to as a Layer 7(of the OSI model) DDoS attack, and the main goal of this attack is to bring down the victim's resources to stop and refuse services. This type of attack is launched in the OSI layer in which the websites are developed and the response to the HTTP request coming from the Internet will be distributed.

The attacks mainly target web servers, and even though their requests look like legitimate demands, they immobilize the server services. Basically, this type of attack is tough to discover because it is very hard to distinguish between normal and abnormal network traffic, therefore detecting and preventing this type of attack is very hard and challenging. The most effective of this type of attack is on bandwidth consumption and in this category HTTP flood and Slow Posts can be mentioned as famous attacks. HTTP Flood: In this attack technique, the HTTP GET or POST request is simultaneously sent from several hosts to the target server, causing the disruption of the server services since the server continues to respond to the requests and eventually would be run out of resources.

3.7.1.3 Protocol attacks

In this type of attack, the vulnerabilities in Layers 3 and 4 of the OSI model are used to exploit an inaccessible victim. Extreme usage of the target resources or network devices such as

firewalls and load balancers can cause an interrupt in their services. Of the most widespread type of this kind of attack is SYN Flood.

TCP-SYN Flood:

Since TCP is a connection-oriented protocol, before sending the data flow between source and destination, communication should be established in the network. Therefore, the attacker can exploit the vulnerabilities of the TCP protocol. In TCP communication protocol, regardless of the legitimacy of the sender, the server has the obligation to respond to all received SYN requests. Before establishing the connection, the server must keep the received requests in its memory and wait for receiving confirmation from the recipient [116].

An attacker can take advantage of the TCP communication process and send an enormous number of SYN requests to the server without the purpose of establishing a real connection (they usually use the non-existent IPs for connection). Hence, by creating several open connections, the server's memory is saturated, making it incapable of serving any more requests, refusing access to the normal users, and the service is disrupted, this attack is known as SYN flooding [117]. In the TCP protocol, to establish a connection between two computers, three negotiation steps (Three-way Hand-shake) must be done (Figure 3.12), in which the client tells the server that it wants to establish a connection by sending an SYN packet, the server responds with an SYN-ACK packet and the client ends the negotiation with an ACK.

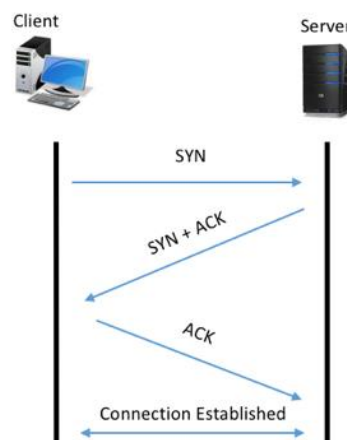


Figure 3. 12: Three-step negotiation in TCP [118].

So, in this attack, the attacker tries to take advantage of the TCP hand-shake concept by keeping the connection open. The attacker sends TCP connection requests (SYN messages) to the targeted server using fake IP addresses, in this case, the server responds by sending an SYN+ACK and waits for the sender to respond with the ACK in return. However, the server will not receive any ACK confirmation since the source addresses are fake, and the server must keep the connection open for the response and continues to send SYN+ACK packets to the sender until a timeout occurs (Figure 3.13).

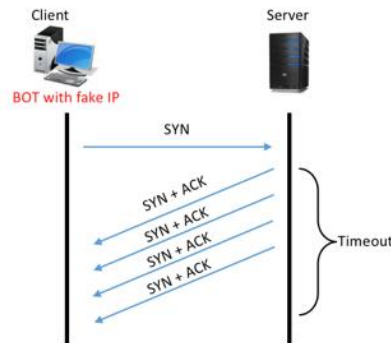


Figure 3. 13: TCP Flood attack. [118].

Some of the other prevalent examples of this type of attack are as follows:

UDP Flood:

In this method, the attacker uses multiple clients for sending UDP packets to different ports of a targeted server. If the requested port is not used for any services or the server cannot resolve the requests, as a response to the sender of the request, it will send an ICMP response with the appropriate message which is the destination is not reachable. This type of request sends to the passive ports from a group of infected computers (Zombie) causes network connection bandwidth consumption. ICMP Flood: This technique uses a similar method as the UDP flood attack, which means a huge size of ICMP requests are sent to the targeted server to pretend the victim to resolve the incoming requests.

ICMP Flooding:

The ICMP protocol is used by some network troubleshooting tools to assist in the identification of network connection errors. The network tools use the ICMP protocol messages to verify the existence of network connection errors. Therefore, not only the ICMP message is continuously sent but also inside of the switch has priority for processing. This feature gives the attacker an opportunity to generate an ICMP flood attack with a large amount of traffic. This activity causes a denial of service since it occupies all the available bandwidth [116].

The ICMP flooding attack has several variations such as SMURF and Ping of Death. The SMURF attack uses ICMP broadcast messages to generate a flooding attack and forces a large number of hosts to respond. If the broadcast message is sent on a fast and continuous basis, the target host will be overwhelmed by processing the traffic and a denial of service attack occurs. The Ping of Death attack uses ICMP messages in the form of echo and request/reply. The attacker generates flooding of packets with a large extension (more than 64Kb) so that the target system cannot process them and as a result, a denial of service attack happens. DNS Flood: In this attack, to disrupt DNS server operation, various name resolving requests are sent to a company's DNS server to overload and suspend its service.

3.7.2 DDoS Attacks on SDN Networks

The DDoS attack is a type of attack directed by multiple computers called “bots” or “zombies”, which are a network of different computers remotely controlled by an attacker to launch massive unwanted network traffic toward a specific target; the objective can be a computer, server or a network [119]. The major goal behind this attack is trying to exhaust the network resources, with the intention to hamper or disrupt the services for normal users. Regardless of the target of the attack, all different types of DDoS attacks have the common characteristic, which is flooding the targeting network with huge amounts of unwanted packets, typically using the following protocols: ICMP, TCP, or UDP packets [120].

As mentioned earlier, during the DDoS attack, to cause the unavailability of service, a vast number of packets are sent to one or more servers in the network. In the SDN network, if the source IP addresses of incoming packets are spoofed, the SDN-based switch will not find the proper match in its flow table and will have to forward the packet to the controller. Therefore, the accumulation of the incoming packets from both the legitimate users and fake packets generated by the attacker can compromise the critical resources such as CPU, memory, and network bandwidth of the SDN controller, and in the worst case, completely disrupt them and eventually bring down the controller (Figure 3.14) [121].

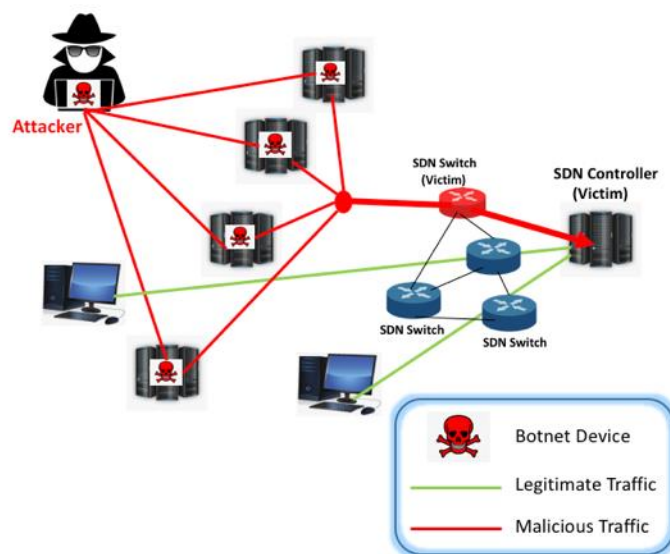


Figure 3. 14: DDoS Attack on SDN Network Using the Botnet [122].

The DDoS attack can be performed by targeting all three planes of the SDN architecture. Therefore, In the SDN environment, DDoS attacks can be divided into three categories based on these potential targets: application-layer DDoS attacks, control layer DDoS attacks, and data layer DDoS attacks. In the SDN network, when the DDoS attack happens, it can extend either vertically when one layer communicates with another layer or horizontally because of communication between controllers and between applications. The DDoS attacker can send a large amount of malicious traffic to any SDN layers through the defined communication channels such as the northbound, southbound, and east/westbound interfaces. The attack can be propagated from the data plane into the control layer through the southbound

interface. On the other hand, at the control layer, the attack can be spread from one controller to another via the east-westbound API. And lastly, the traffic from the control layer can be directed to the application layer through northbound API to paralyze different services [122].

3.7.2.1 DDoS attacks at the SDN application layer

In this situation, the attacker sends malicious traffic to the application layer in order to saturate the northbound interfaces and disrupt running applications. Since it is tough to identify traffic between different SDN applications, it is possible for a DDoS attack targeting to influence another application that is not the attacker's target [119]. More specifically, in this layer, these types of attacks establish full TCP connections targeting network entities such as servers and then start flooding the illegitimate traffic generated by the attacker towards it with a huge number of HTTP requests to saturate the bandwidth. It is important to mention that, in the case of the attack performing at the start with a low and slow rate, it would be very challenging to distinguish them from legitimate traffic. Therefore, at present, the tools for SDN application layer attacks are a powerful facility for attackers to harm their victims. The most important challenge about this security issue which should be considered is differentiating between malicious traffic and an unexpectedly large volume of packets from legitimate users [123].

3.7.2.2 DDoS attacks on the SDN data layer

At the data layer, the appropriate target for the DDoS attack would be the OpenFlow-based switches and their flow table, because they contain information such as communication, access control, and administrative data. In this case, through unauthorized physical or virtual network access the intruder intentions to disrupt network functionality. Since the storage capacity of the OpenFlow-based switch is limited, it is not possible to store all the flow rules. Therefore, if the attacker conducts a vast number of packets from an unknown different IP address in a short period of time, the switch considers them as new flows and sends them to the controller, then the controller creates new rules for these new flows, forwards and insert them into the flow table. In this case, in a short period, the capacity of the flow table gets full and runs out of space. Therefore, it is not possible to add a new rule to the flow table anymore. Accordingly, since the flow table is out of capacity, the transmission of new legitimate traffic from the switch to the controller is stopped [119].

3.7.2.3 DDoS attacks at the SDN control layer

Since the SDN control layer is a centralized entity in the SDN architecture that controls the entire infrastructure network. Therefore, attacking the SDN controller using DDoS attacks can cause bringing down the entire network and unavailability of the network services by sending huge amounts of traffic from multiple sources to overload the controller.

As the SDN concepts, for every new coming packet, the decision-making is done by the controller. In the data plane, when a network packet arrives at the switch and the switch cannot find any flow rules corresponding to this packet in the flow table, for making the decision, the entire packet or part of the header is sent to the controller. Therefore, sending a total packet in a high volume of network traffic from the switch to the controller can saturate the southbound interface bandwidth [119].

To perform such a kind of attack, the attacker in the first step must identify if the targeted network is SDN-based. It should be noticed that in traditional network architecture, the network devices often have a pre-defined forwarding table. Therefore, for a new incoming packet, no additional time is required to process, create and add a flow entry. But, in SDN architecture, for a new incoming packet, the controller needs some time to create a new flow entry. In addition, more process time should be spent comparing the first packet with subsequent packets from the controller. Therefore, this knowledge can be used by attackers to identify whether the target network is SDN or not. It means that the attacker identifies whether the target network is SDN by checking the difference in controller response times between the first and subsequent packets and once he finds out that the network is SDN, the attack can be performed [124].

3.8 Chapter Summary

The major goal of this chapter is to answer the first research question which is:

What are the SDN attack vectors?

To achieve this purpose, this chapter, there has presented the SDN environment in detail. First, introduce SDN history and concept, then define the SDN architecture deeply. More specifically, we study Open Flow protocol as the southbound protocol, because some traditional protocols can be considered as southbound protocols in SDN environments such as SNMP, NETCONF, LISP, and even BGP. In addition, there are also several other specific Southbound APIs to manage communications between these two layers of the SDN architecture, for instance, P4, OVSDB, OpFlex, and ForCES, but OpenFlow is the most famous Southbound API and even it is considered the industry standard for controlling and managing the overall information about the network equipment, Therefore, in this research, we consider the OpenFlow protocol as the southbound interface for the SDN networks to provide the communication channel between SDN-based switches and the controller. Furthermore, we explore the SDN security on each layer of the SDN architecture to be able to identify the SDN attack vectors.

Moreover, DDoS attacks have been studied, because these types of attacks are a real threat to both traditional and SDN networks. This type of attack can cause a large disruption to any network infrastructure. Particularly, in SDN networks due to the separation of the control and data plane, the DDoS attack can be the most important threat vector for SDN networks.

Chapter 4

4 Evaluating the SDN Controllers

It is essential to have a solid understanding of the performance of the control plane in software-defined networks (SDNs) since this plane serves as the "brain" of the network and influences the overall functioning of the network, as well as the performance of services, applications, and so on. Therefore, for developers, experts, and users to make efficient use of this technology, it will be helpful to have a solid grasp of knowledge about the performance of the SDN controllers and evaluate different controllers already available for SDN networks in the research and industry domain. There are several SDN controllers available in the market, in this research, we consider the most prevalent open-source controllers, such as NOX/POX, Ryu, Floodlight, ODL(OpenDaylight), and ONOS, and for evaluation of these controllers we deliberate the following features:

4.1 Features for Selecting the SDN Controllers

In this section we introduce the features we consider for evaluating and comparing various SDN controllers:

1. (South-, Northbound) Interface support: The interface communication channels the way that the different layers of the SDN can communicate with each other. When choosing a controller, it is essential to understand which interface communication protocols are supported by the controller. From the southbound interface point of view, the controller must be capable of supporting different protocols beyond just the OpenFlow, such as NetConf, P4, and OF-Config. Since OpenFlow protocol is a standard southbound interface that is used specifically in the research environment for this research we consider OpenFlow protocol. Therefore, the characteristics of the OpenFlow versions that the controller supports should be considered, as well as the possibilities offered by the controller provider to migrate to new versions of the protocol, such as v1.4 and v1.5. Because some important features, such as IPv6 support, are not part of OpenFlow v1.0 specification as they are added in the OpenFlow v1.2 standard. For the Northbound interface protocol, there are various methods offered by different vendors, such as RESTful API and gRPC. In this research, we use REST API to provide communication between the controller and the detection module.

2. Performance: We consider two features to define the performance of the SDN controllers. One is the number of flows that can be processed per second by the controller and the other is flow shaping because the major function of the SDN controller is to make decisions about flows. When switches initiate more flows than the capacity of the controller these metrics significantly influence. As mentioned earlier, there are two methods to handle the flows: proactive and reactive. The Proactive method is when decision-making is done before the network packet reaches the SDN-based switch. Thus, when the packet reaches the switch, it informs about the decision and what to do with the packet. This method significantly reduces

the delays and provides no real limit on the number of flows per second that the controller can support. Reactive method, the flow handling process occurs when the SDN-based switch receives a packet that does not match any entry in its flow table, and therefore the packet should be forwarded to the controller, for decision-making. In the reactive method, the consuming time for processing the packet can be defined as the calculation of the time it spends to transfer the packet from the SDN-based switch to the SDN controller, the total time of the flow processing on the controller, and the time it spends to fill the flow table on the switch. In both methods, the key factors that influence the time of the flow processing contain the processing capacity of the SDN-based switches, the southbound interface bandwidth, and the input/output processing power and throughput of the controller.

3. Architecture: For evaluating various SDN controllers, it is vital to consider the architecture and design of the controllers. From the architectural point of view, there are two SDN controller categories: A centralized design, securely connected control plane, and distributed, and scalable SDN controllers. In the centralized architecture, the implementation and maintenance would be easy and it can provide low latency between different layers of the SDN architecture but centralized design can be a bottleneck. On the other hand, in the distributed architecture, the implementation and maintenance are mostly more sophisticated but provide an environment that can be scaled efficiently to prevent performance bottlenecks.

4. Scalability: It is a fundamental consideration feature for comparing various SDN controllers. When evaluating SDN controllers, it is necessary to ensure that the controller is capable of reducing the impact of heavy network traffic overhead. One aspect of scalability can be defined as the ability of the SDN controller to span multiple locations. To maximize the benefit of this capability, the SDN controller must permit network routing and forwarding policies to be applied automatically in different locations. Another aspect of scalability can be the capability of the controllers for internal functionality for defining and supporting the clustering design.

5. Reliability: The two significant factors which indicate the reliability of the SDN controllers are elasticity and fault tolerance. There are various techniques for providing consistency to the controllers. For instance, one of the techniques that an SDN controller can use to increase network reliability is the ability to discover multiple paths from source to destination, which it can be done by constantly monitoring the network topology to prevent packet loss conditions in case of interruption of the link between source and destination. Alternatively, if the SDN controller distinguishes only a single path from source to destination, it should be capable of utilizing a mechanism to rapidly react when a link failure occurs, and redirect network traffic to an active link.

Another technique for providing high availability and fault tolerance for the controller itself is, there should be a built-in mechanism in the controller for clustering which means it should be capable of creating various, identical but related nodes. With this feature, in case of a controller failure, another instance can take the role, control, and maintain the network. In the case of running one single controller, there should be an external node monitoring the running controller to discover and respond if any failure occurs. In addition, it is vital that the controller has a mechanism for providing redundancy for both hardware and software.

6. Monitoring and Visualization: One of the important features which must be considered for comparing SDN controllers is the ability to monitor and visualize the SDN network topology to discover and present the physical link between network entities. This would help the network administrator to have not only the overall view of the network topology but also provide the ability to rapidly discover any link failure between network devices. In addition, the controller must be capable of providing a view with detailed information about the flows, both from the physical and virtual network perspective. To achieve the above-mentioned capability, the controller must be able to continuously monitor the network topology with the help of some standard management protocols for instance SNMP (Simple Network Management Protocol), and provide access to network information such as port or link status, connected device, etc., using a common communication protocol such as REST API or gRPC.

7. Support Community: In recent years, the industry and business domain have shown growing interest in the SDN concept, numerous providers have entered the SDN market, and many others have declared their objective to do so. Organizations and societies who are responsible to evaluate SDN controllers should concentrate not only on the aforementioned SDN controller's technical features but also on the vendor's support. Because the SDN market in general, and the SDN controller market, in particular, are still unpredictable.

For every organization that intends to deploy SDN architecture in their network, it is critical to purchase the SDN drivers from a vendor that can guarantee to support its product and keep up with the changing in the SDN environment. Most of the well-known and prevalent SDN controllers run on an open-source platform, therefore it is significantly beneficial if a large developer and user are involved in the development of these projects. Involving a large international community can provide long-term support and security to SDN projects.

There are other features that can be considered to evaluate and compare SDN controllers, such as Network Programming, Modularity and Extensibility, Network Virtualization.

4.2 SDN Controllers

The primary OpenFlow-based controllers that are now available and most often used are as follows:

4.2.1 NOX/POX:

1- NOX Controller:

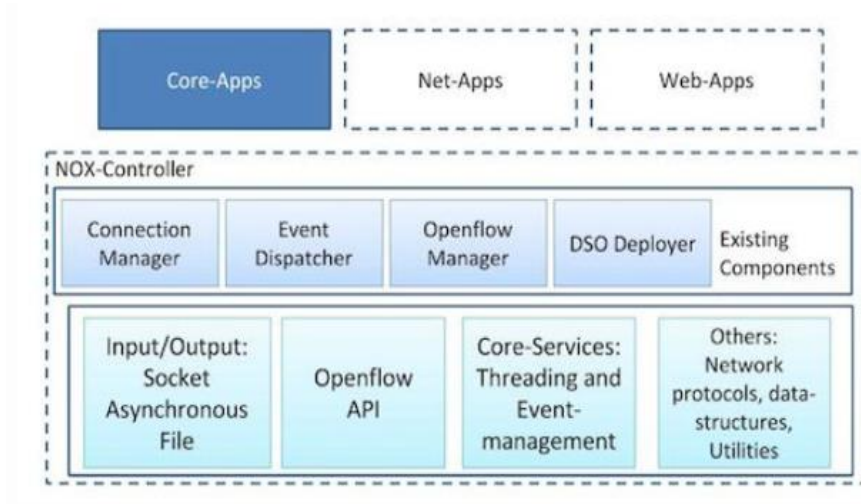


Figure 4. 1: NOX Controller Architecture [Taken from [125]]

NOX⁷ is an open-source project that is extensively used and stable. It is the first SDN controller that was developed to support OpenFlow. This controller, in the beginning, was called classic NOX; this initial version was created using Python and C++ programming languages; however, currently, it is no longer in use. The latest version is entirely written in C++, which makes it significantly quicker; also, it is maintained by numerous teams, and they still provide updates for it. Comparable to most controllers, this controller operates by monitoring events and providing a platform for programming a set of activities that must be performed prior to each event. NOX is suggested to be programmed by those who know C++ programming and prefer to employ instructions that are not too complicated; in addition, because of its simplicity, it often produces excellent results in terms of performance. Figure 4.1 shows the overall architecture of the NOX controller.

2- POX controller

POX⁸ is essentially a Python-based implementation of NOX, with the limitation that it only supports OpenFlow version 1.0, which can be considered its drawback and is not as fast as the version that is developed in the C++ programming language.

Since POX is continually being updated, this is the reason that it is still widely utilized, and it is also very straightforward to understand the written code and develop it, so for those who are familiar with Python programming, this controller is extremely suggested. Additionally, it

⁷ "NOX". [Online]. Available: <http://www.noxrepo.org/>, [Last accessed: June 2022].

⁸ "POX Manual Current documentation(2015)". [Online]. Available: <https://noxrepo.github.io/pox-doc/html/>, [Last accessed: June 2022].

enables rapid programming, which makes it an appropriate option for presentations, tryouts, and scientific investigations.

4.2.2 Ryu Controller

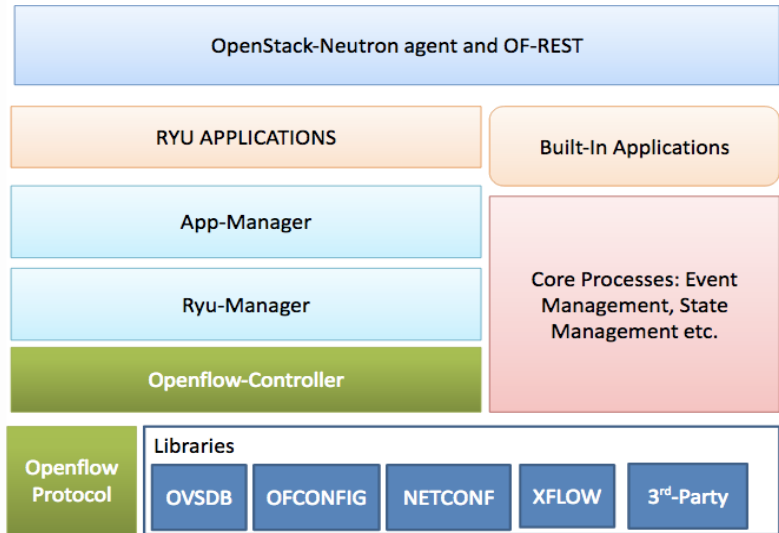


Figure 4. 2: Ryu Controller Architecture [Taken from [126]]

Ryu ⁹ is an open-source software and component-based application. It is developed completely in Python and maintained by NTT’s labs. Ryo is comparable to other SDN controller frameworks that can offer developers the ability to generate new control applications and network management by using software components with well-defined Application Programming interfaces (APIs). One of the powerful features of Ryu is that it can maintain and employ various southbound protocols for managing network devices. These protocols include such as Configuration Protocol (OF-Config), Network Configuration Protocol (NETCONF), OpenFlow, and others. In addition to the aforementioned protocol, Ryo also supports Netflow, OF-Config, sFlow, OVSDDB, and other third-party protocols.

Furthermore, the OpenFlow controller is one of the essential elements that make up the Ryu architecture. This component is in charge of controlling the OpenFlow switches, which are used to set up flows, handle events, and so on. (Figure 4.2 illustrates the architecture of the Ryu). Ryu supports this protocol up to the latest version 1.4.

⁹ “Ryo Controller”, [Online]. Available:” <https://ryu-sdn.org/>, [Last accessed: June 2022].

4.2.3 Floodlight Controller

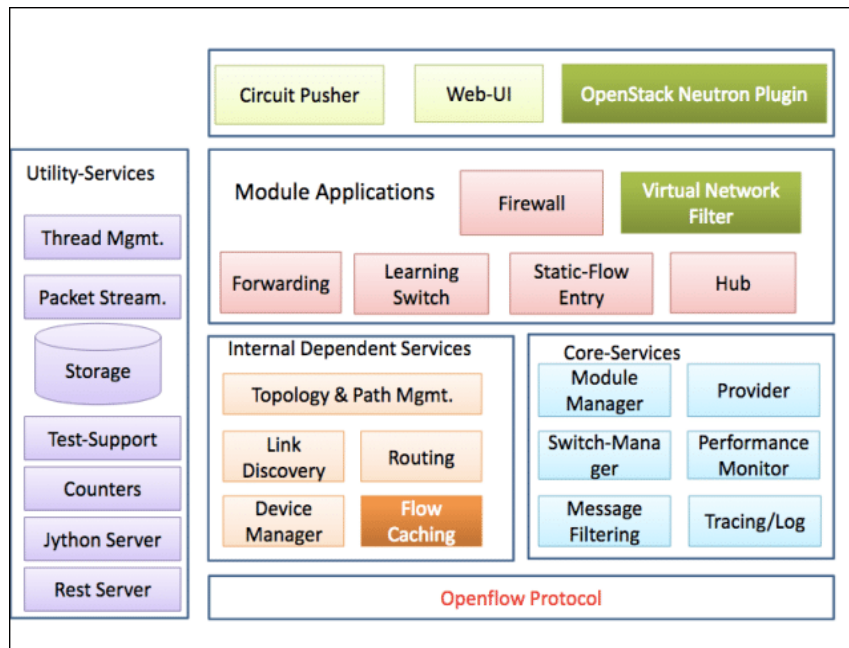


Figure 4. 3: Floodlight Controller Architecture [Taken from [127]]

Floodlight¹⁰ is a Java-based open-source controller that is compatible with OpenFlow. It was developed by a community of Big Switch Networks developers and, originally, would be included in the OpenDaylight project. Apart from the fact that it is developed in the Java programming language, this controller differs from the preceding controllers in that it was the only one that implemented a REST API [128] for the Northbound interface at that time, which offers a possibility for the incorporation of external apps(Figure 4.3 shows the Floodlight architecture).

4.2.4 OpenDaylight (ODL)

OpenDaylight (ODL)¹¹ is a joint open-source project founded by the Linux Foundation that claims to be the most extensively used open-source SDN controller today, according to its official website. Currently, several SDN controllers have been developed, the most used controller in the real world are ODL (OpenDaylight) and ONOS. The essential purpose of ODL is to simplify the deployment of the SDN and Network Functions Virtualization (NFV) technologies in existing conventional networks. Its main characteristics are:

¹⁰ Floodlight, [Online]. Available: <https://floodlight.gitlab.io/>, [Last accessed: June 2022].

¹¹ "The OpenDaylight platform", [Online]. Available: <https://www.opendaylight.org/>[Last accessed: June 2022].

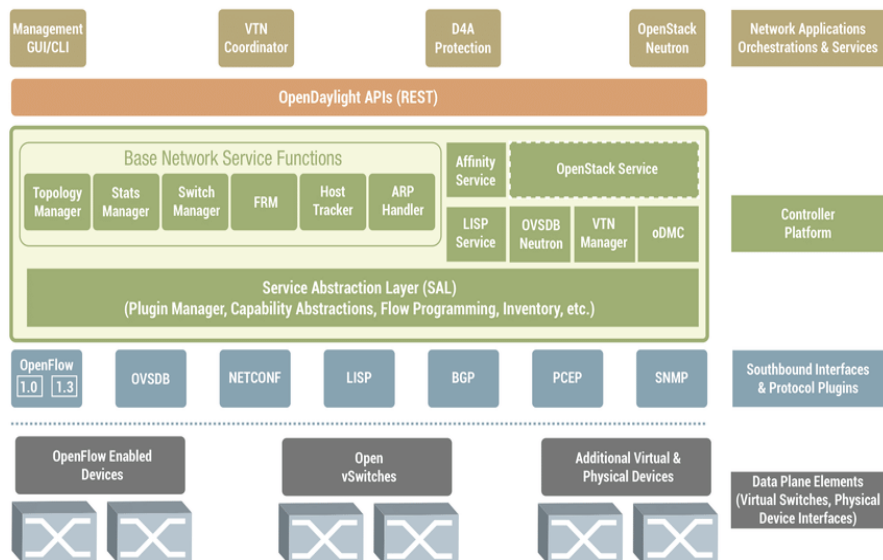


Figure 4. 4: ODL Controller Architecture [Taken from [129]].

It provides a modular, extensible, scalable, high availability, and multi-protocol infrastructure. ODL is capable to support various southbound protocols, such as OpenFlow, NETCONF, and OVSDB. It is written in Java and uses the YANG language for data modeling, although the communication between the different interfaces is accomplished through REST API. Several important telecommunications companies provide support for this project. Cisco, HP, and Intel are just a few prominent companies that are members of the ODL project (Figure 4.4 presents the ODL architecture).

Phosphorus was released in September 2021 and was the most recent release at the time of writing this article. The main objective of this emerging paradigm for network orchestration is to make the architecture of a network more dynamic, easier to reconfigure, and compatible, independent of the models or manufacturers used to construct the network [130]. By abstracting high-level operations from the devices, it allows operators to maintain, control and manage all network services from a distance. By separating the control plane from the data plane in the network, the purpose mentioned above can be achieved. In this methodology, the control layer of the network devices is separated from the hardware, and decisions making, and policies defining the packet routing and traffic flows will be done outside and independently of the network devices that exist in the data plane. The following are some of the advantages afforded by the implementation of SDN:

- Network automation and streamlining are accomplished by simplifying the processes that must be performed on it.
- As an accomplishment, the complexity is reduced through the separation of the control plane and the data plane.
- By creating a programmable network, the manual configuration will be eliminated.
- Using this approach will be led to a more secure and scalable network since human interference is decreased.

-The usage of an Application Programming Interface (API) in this architecture causes speeds up the development and deployment of new applications and services.

4.2.5 ONOS Controller

Open Network Operating System (ONOS) [131] is a central controller for managing network components in an SDN environment. This operating system was first introduced by the Open Networking Lab (ON. Lab) in 2014 and currently is maintained and developed under the umbrella of the ONF (Open. Network Foundation).

ONOS is an open distributed operating system and an SDN controller platform for providing high scalability and availability. ONOS’s major advantage is supporting multiple protocols across southbound interfaces to communicate with different devices and offering APIs as northbound interfaces to address the service providers’ and application developers' requirements (Figure 4.5). From the first version of the ONOS, which was released in December 2014, till now, several versions of the ONOS has been released, and the latest one is X-Wing which was released in July 2021 [132].

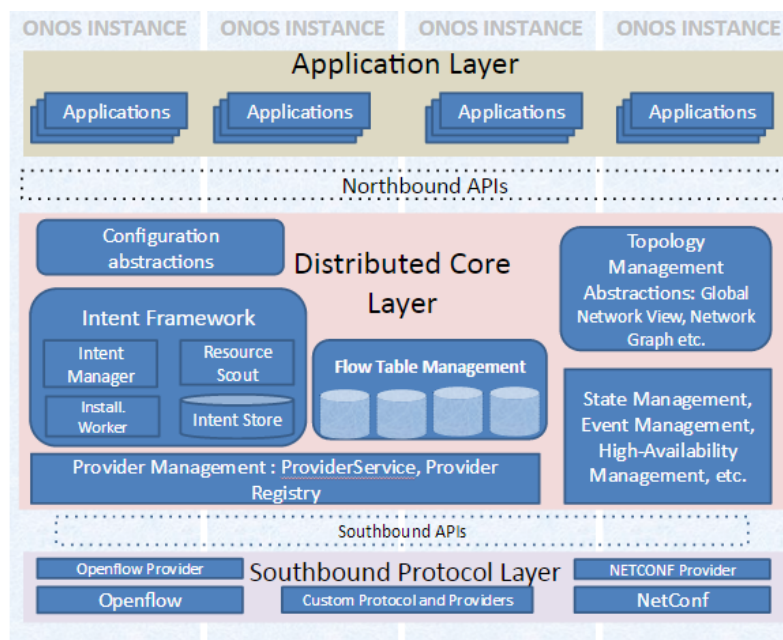


Figure 4. 5: Overview of the ONOS Architecture [Taken from [133]]

4.2.6 Conclusion

We investigate some of the prevalent SDN controllers, the table 4.1 summarize the result, and based on the features we consider to evaluate SDN controllers, for this research, we choose ONOS as the central controller in our proposed framework. Therefore, we study deeply through the ONOS characterization, structure, and architecture in the next section.

Table 4. 1: SDN controllers and their characteristics [134].

Controller	Programming Language	GUI	Documentation	Distributed/Centralized	Platform	Southbound API	Northbound API	Partner	Domain
ONOS	Java	Web-based	Good	D	Linux, Mac, and Windows	OF1.0, 1.3 NETCONF	REST API	ON.LAB, AT&T, Ciena, Cisco, Ericsson Fujitsu, Huawei, Intel, NEC SK Telecom	Datacenter, WAN, and Transport
OpenDaylight	Java	Web-based	Very Good	D	Linux, Mac, and Windows	OF 1.0, 1.3, 1.4 NETCONF YANG OVSDB PCEP BGP/LS LIST, SNMP	REST API	Linux Foundation with Memberships covering over 40 companies such as Cisco, NEC, IBM	Datacenter
NOX	C++	PyQT4	Poor	C	Linux, Mac, and Windows	OF 1.0	REST API	Nicria	Campus
POX	Python	PyQT4	Poor	C	Linux, Mac, and Windows	OF 1.0	REST API	Nicria	Campus
Floodlight	Java	Web/Java based	Good	C	Linux, Mac, and Windows	OF 1.0, 1.3	REST API	Big Switch Networks	Campus
Ryu	Python	Yes	Fair	C	Linux	OF 1.0, 1.2, 1.3, 1.4 NETCONF OF-Config	REST API	Nippo Telegraph And Telephone Corporation	Campus

4.3 Open Network Operating System (ONOS)

ONOS [135] is one of the most popular applications that is used to provide the role of the controller in an SDN environment. Its primary goal is to make the implementation and development of SDN and NFV solutions as simple as possible. In addition, it is an open-source system and several well-known high-technology companies such as AT&T, Huawei, Intel, etc. support it. Although some essential characteristics are shared between both controllers.

The following are the distinctive characteristics of ONOS in comparison to ODL:

Using Apache Karaf makes it possible to activate or deactivate any of ONOS's functionalities while the system is running. This is quite beneficial for expanding the controller's capabilities without the necessity to stop and restart the service. These characteristics make it easier to create and develop new applications and solutions in the future. Because of this capability, ONOS may be used by a wide range of manufacturers and different types of network equipment. Its major purpose is to make it possible for service providers to implement actual SDN/NFV solutions.

Using this system in SDN environments has the following advantages:

- Optimal performance for the ONOS platform and applications as a modular, extensible, and distributed controller.
- Provides significant ease of network management, since it considers the entire network as a single entity.
- Runs as a distribution system between multiple servers while offering fault tolerance in the case of a server failure, it permits the utilization of CPU and memory from several servers.
- Provides some types of functionality comparable to server operating systems, such as resource allocation and permissions, user-interactive tools such as CLI and GUI, and APIs and abstractions.
- Provides a network graph and a view of the entire network in an abstract manner, independent of the northern part, which is logically centralized despite the physical distribution between several servers.
- Establish a control platform for service providers to address the problems of distributed systems, including high availability, scalability, and efficiency.
- The greatest benefit of an operating system is that it offers a functional and useable foundation for software applications intended for a certain purpose or use case, including custom communication routing or management and service monitoring for software-based networks.

4.3.1 ONOS Use Cases

The use cases are intended to address challenging SDN deployment scenarios encountered by members of the ONOS community. Correspondingly, the implementations of these use cases are typically complicated, involving combinations of apps, drivers, system core components, and specialized APIs. In the following, the names of some of the applications as use cases for ONOS are mentioned. These applications are developed on the ONOS, or integrate ONOS as part of their software design [136]:

- CORD: Central Office Reimagined as a Datacenter (vCPE, vOLT, NFaaS)
- CORD: Leaf-Spine Fabric with Segment Routing
- DC Network Virtualization
- E-CORD: Enterprise CORD
- Packet Optical Convergence
- IP RAN
- M-CORD: Mobile CORD
- NFV (NFaaS)
- Peering Router - ONF's Project Atrium
- SDN-IP
- Virtual Private LAN Service (VPLS)

The above-mentioned use cases have their own website which contains project definition and architecture documents, related presentations, and videos.

4.3.2 Clustering capability of the ONOS

One of the significant benefits of using ONOS as a controller in the SDN environment is in ONOS; unlike other controllers, the support for distributed architecture was considered in its initial design. ONOS can be developed as a set of controllers that operates together to achieve flexibility, fault tolerance, and better load management. ONOS has the ability to form a cluster. Each cluster consists of controllers, which are called nodes, and each node can be verified by a unique identifier. There are three states in which one node in the cluster can have:

None: in this state, the node cannot communicate with the network elements.

Master: It is the state when the node has complete control of the network elements which means that the node has the ability to read its state and execute write operations on it.

Standby: the node in this state has a connection with the network elements, but it has only permission to execute read operations from the network element.

In the beginning, when a cluster is established and all nodes are started, all of them will be in the “None” state. When there is no master node in the cluster, the first node detects a network element and makes a connection to it, it will become the master node.

There is a subsystem leader in the cluster who is responsible for ensuring that each network device has only one master node(controller) and all other controllers will remain in the standby state. Also, in the cluster, the standby controllers are prioritized to ensure that control of the connected network device can be quickly restored in the case of a failure of the master controller. Regarding cluster synchronization, each ONOS controller is aware of a subset of network information and shares it among other members of the cluster as events. These events are controlled and maintained by the controller that manages that information.

In a cluster environment (with multiple controller instances), various failures may occur such as the crashing of one node(controller), In this situation, there should be an approach that the other node in the cluster is able to receive updates from others. In this case, ONOS proposes an approach for instance the anti-entropy mechanism, which is based on the intervallic node exploration and gossip protocol.

4.3.3 Security concerns in the ONOS controller

Based on the research in terms of the security and performance analysis of the ONOS controller [137], some security points must be considered in the ONOS controller. These security concerns can be divided based on different methods of communication:

- In Northbound Interface: The standard HTTP protocol is used by default to allow access between the control plane and the application plane. Despite the fact that authentication is set by default, there is no security in the data transmission between the client and the ONOS server if HTTPS is not utilized. Because of potential compatibility issues with specific browsers and REST APIs, it is not recommended to use self-signed certificates.

- In Southbound Interface: By default, the data transfer through the southbound interface is not encrypted, and the devices using this channel are not authenticated. Therefore, it is suggested that SSL protocol be enabled in the southbound interface to avoid data eavesdropping and tampering. Moreover, when the ONOS server and network devices have been linked, it is recommended to enable TLS protocol.

- In East/West Interface: By default, for communication between the controllers through the East/Westbound interface, TLS is not enabled. Consequently, for this kind of communication, it is highly recommended to use TLS.

Since ONOS has been chosen to be the central controller of the proposed framework for this research. More details about its architecture in detail will be explained in the coming section.

In this chapter, the architecture of the ONOS controller will be explained in detail, and clarify the different methods which are used at the user level to interact with it.

4.3.4 ONOS Overview

There are several ways and tools which can be used for installing the ONOS controller, in this research, we use the Bazel tool for installing the ONOS controller.

Note: Bazel is an open-source tool for building and testing applications similar to Make, Maven, and Gradle. It is designed to support the projects that is developed using multiple languages and construct outputs for several platforms. Bazel can accommodate big codebases that are spread out over numerous repositories in addition to a high number of users.

After completing the process of the ONOS installation. The following ways can be used to communicate with the controller.

4.3.4.1 Communication with ONOS

There are three ways in which a network operator can interact with ONOS:

The first way is using the CLI console [138]. This console is developed and employed based on Karaf which the network operator can directly enter the command on the running ONOS server. Figure 4.6 shows the ONOs CLI interface.

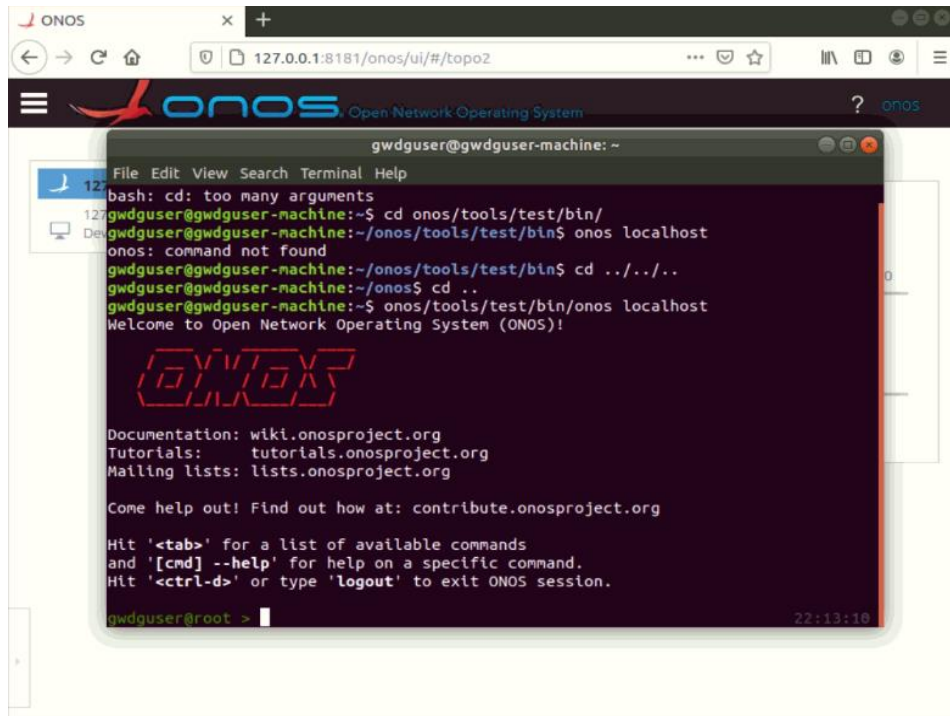


Figure 4. 6: ONOS CLI interface

The second way is using Graphical User Interface (GUI) [139] through the web browser that provides a visual view of the network which is controlled and maintained by the ONOS (Figure 4.7).

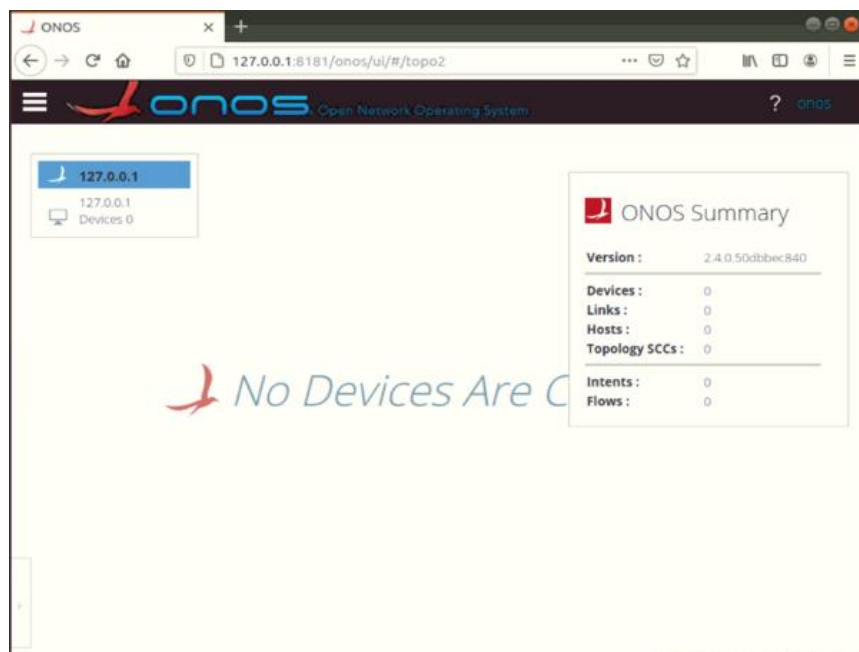


Figure 4. 7: ONOS GUI

The third way is using the REST API which allows interaction through the REST protocol with the interface provided by ONOS.

The ONOS server provided an extensive REST API for interaction using the REST protocol. This protocol proposes the same functionalities similar to the CLI console by using the CRUD requests (The CRUD stands for the four main requests create, read, update and delete). The ONOS REST API can be accessed at the following link:

`http://[ONOS_SERVER_IP_ADDRESS]:[ONOS_PORT]/onos/v1/docs`

4.3.5 ONOS Architecture

The ONOS kernel and its central services are written in Java as bundles, which are loaded into the Karaf OSGi container. OSGi [140] is a module system for Java that enables the installation and dynamic running of modules in a JVM (Figure 4.8).

OSGi is a dynamic component platform for creating and distributing modular Java libraries and software applications, to make them easier for developing code, make application distribution more manageable, detect any issues fast, gain more reusability, and provide better vision into the application during the run time.

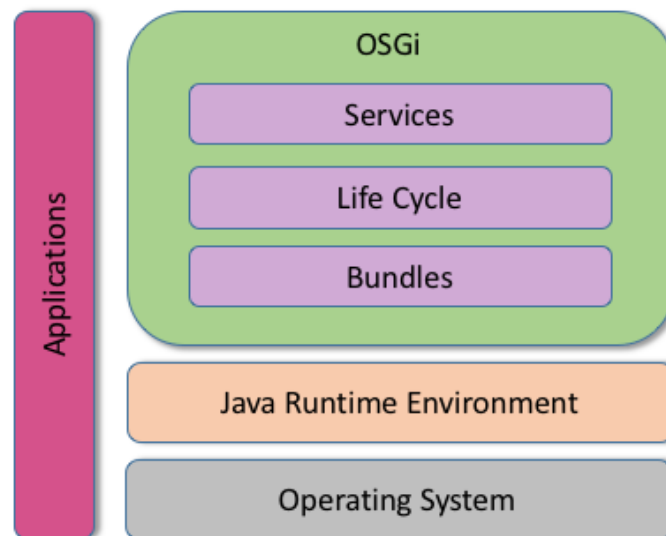


Figure 4. 8: Overview of the OSGi framework [Taken from [141]].

ONOS contains several high-availability databases, with reliable interactions, scalability, and different approaches for performance improvement such as replication techniques, robust compatibility, and as well as using gossip protocol for synchronization across multiple controllers and for maintaining its cluster members. In addition, it uses Hazelcast, a Java-based open-source in-memory data grid for facilitating cluster membership. From ONOS version 1.4 the Hazelcast has been replaced by the Atomix framework [142]. This framework is based on

an algorithm known as the RAFT consensus algorithm to provide a better solution for managing recovery.

The following are the fundamental concepts of the ONOS design:

- High availability, scalability, and performance.
- Simplicity and solid abstraction.
- The protocol and the equipment's performance are independent.
- Modularity.

Similar to SDN architecture, The ONOS structure is divided into three layers: Southbound Interface, Core, and Northbound Interface. A variety of components and subsystems, which are organized into layers of functionality, make up the ONOS system.

The ONOS architecture (Figure 6.9) is divided into applications (Apps), *Northbound (consumer) API*, *Core*, *Southbound (provider) API*, *Providers*, *Protocols*, and *Network Elements*.

The following figure is a high-level representation of the ONOS architecture.

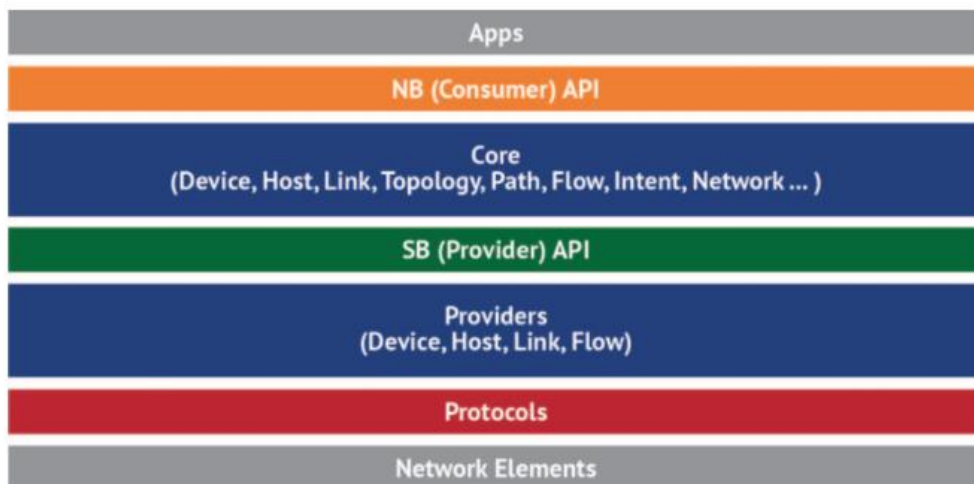


Figure 4. 9: ONOS architecture [143].

To develop an API for ONOS, it is essential to understand the architecture of ONOS and all the services it provides in detail. Therefore, in this section, all above mentioned different parts of the ONOS architecture with the related services they support will be defined.

4.3.5.1 ONOS Application

Through the *AdminService* and *Service* interfaces, applications utilize and change information contributed by *Managers*. Each application is identified by a unique identifier called *ApplicationId*. ONOS uses this identity to manage the context associated with the app. It is

necessary for each application to register to the *CoreService* by specifying its name to receive a valid identifier.

4.3.5.2 ONOS Northbound interface

The Northbound interface [144] consists of components that are responsible for communication between the ONOS core and running services and applications over the network.

4.3.5.3 ONOS southbound interface

The Southbound interface in ONOS is responsible to provide a channel for communication between the core of the control plane and the devices which directly related to the physical network. This interface provides the following feature: Abstraction, modularity, and interoperability. All the components which construct the structure of the ONOS reside in one of the three main ONOS layers which means the Northbound interface, the Core, and the Southbound interface. Figure 4.10 depicts the outline of the relationship between subsystem components in a distributed environment. The dotted lines demarcate the boundaries between the core and the Northbound, and Southbound interfaces.

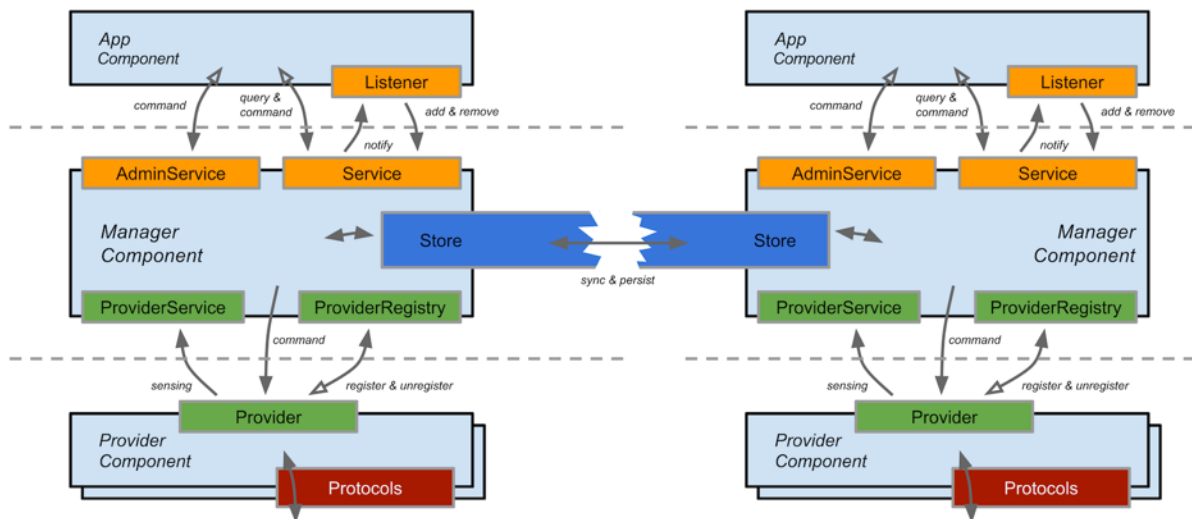


Figure 4. 10: Relationship between ONOS subsystem components [143].

4.3.5.4 ONOS Core

The core [145] of ONOS forms the fundamental of the architecture. It is responsible for tracking all the information which is received about the state of the network and presenting it to the application layer via services.

The ONOS core is composed of multiple services and subsystems as can be perceived from the Figure 4.11. A service is a functional unit that is made of numerous discrete components that together form the stacking software system and the collection of components that compose

a service is called a subsystem. A service consists of multiple components that are spread over different layers. The core provides the following primary services:

Device Subsystem: This subsystem is responsible to manage the list of devices on the network that are participating in the routing tasks.

In the infrastructure network, there are various types of network devices, for instance, switches, routers, or access points and these devices can be identified by a set of specific information such as interfaces/ports and device-id.

Host Subsystem: This subsystem is in charge of managing the list of end *hosts* and their positions on the network. The nodes that have the role of the source and destination points for the network traffic which is called hosts. A host can be a computer. A server, or any other device that generate traffic into the network. A host can be verified by its IP address, MAC address, and connection port.

Link Subsystem: This subsystem is responsible to control the list of *links*. Links mean the direct connection between the nodes. The only types of connections that are permitted in ONOS are those that connect two devices or a device with a host. Direct connections between hosts are not permitted.

Topology Subsystem: This service offers a graph snapshot that presents the whole network that is sorted according to the time. Different routing algorithms such as Dijkstra utilizes this service.

Path Subsystem: This service is responsible to calculate and discover the routes (or paths), which are made up of one or more connections between two network hosts (which must be adjacent). To achieve this goal, this service uses the most recent provided snapshot from Topology Subsystem.

FlowRule Subsystem: It is responsible for providing the flow metrics and measurements as well as managing the flow rules inventory and providing the list of match/action rules which is installed on the devices.

Packet Subsystem: This service provides the possibility for the application to monitor and inject traffic via one or more devices into the network, as well as inspect data packets received by network devices.

The services consist of two entities:

- 1- Manager:** This entity is responsible for communication between southbound interface protocols and the applications.

To achieve this purpose, it provides a number of interfaces that are responsible for providing the current network state for the applications at a particular moment, performing administrative commands on the computers, and creating communication between the manager and the applications.

2- Listener: This entity provides the possibility for applications to register to obtain announcements of events that happened on the network.

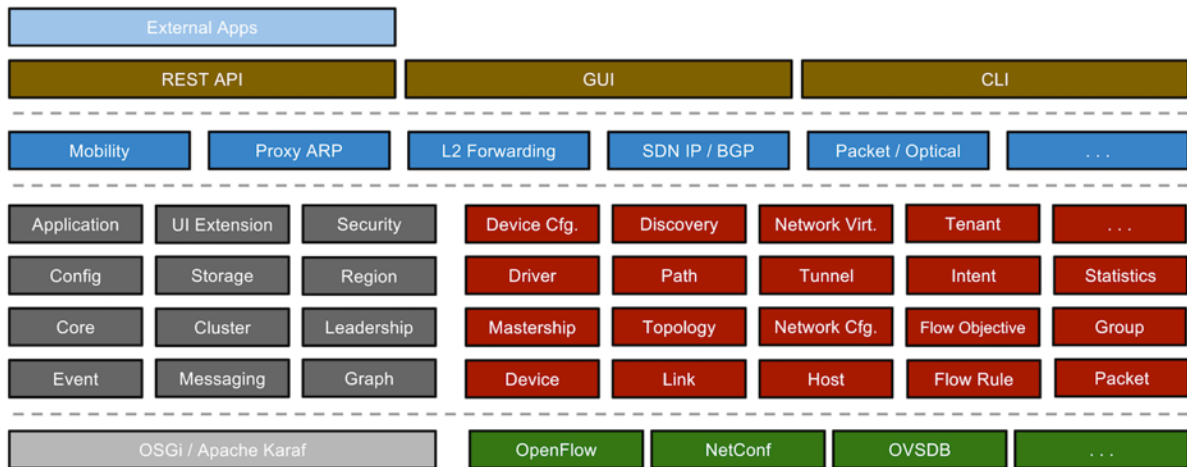


Figure 4. 11: ONOS Subsystem [143].

4.3.5.5 Provider Layer:

In ONOS, the Provider concept is used to cover protocol concerns for the southbound protocols. It also covers any behavior or needs, from other components of the controller platform; The Providers are in charge of providing specific data for the ONOS Core by communicating with the network environment through different control and configuration protocols. In the architecture of the ONOS, the Provider takes Core requests and applies them to the network using the specific protocols [133]. In ONOS The Provider layer is the lowest in the ONOS stack, Providers communicate with the network through specific protocol libraries, and with the Core through the *ProviderService* interface.

1. Manager

It is a component in the ONOS Core. The Manager is in charge of receiving information from *Providers* and distributing it to *Applications*. It contains the following interfaces:

Northbound Service: This interface is used by apps and other Core components to get information about certain features of the network status.

AdminService: this interface is used for executing administrative commands and employing them to the network status in the system.

Southbound ProviderRegistry: By using this interface, Providers may register with the Manager and engage with him.

2. Store Component

ONOS Core has a component called the store, which is linked to the Manager. The Store maintains the information received from the Managers and synchronizes it. This contains promising the reliability and healthiness of the information through the instances of ONOS.

3. Application

Through the AdminService and Service interfaces, applications utilize and change information contributed by Managers.

Each application is identified by a unique identifier (ApplicationId). ONOS uses this identity to manage the context associated with the app.

It is necessary for the applications to register to the CoreService by specifying their name to receive a valid identifier.

4.4 Chapter Summary

In this chapter, there has been investigated different SDN controllers. There are several SDN controllers available in the market, in this research, we consider the most prevalent controllers, such as NOX/POX, Ryu, Floodlight, ODL(OpenDaylight), and ONOS. The main reasons we consider these SDN controllers are they are open-source, the most popular and widely used controllers particularly in the research and scientific realm.

To indicate the SDN controller for our proposed framework, we evaluate the above-mentioned controllers based on the following features: **South and Northbound interface** because once deciding about the controller, it is necessary to understand which interface communication protocols are supported by the controllers since later it should be used to transfer the network traffic between SDN layers in the proposed framework. **Performance**, as in an SDN network, is vital to efficiently manage the process of the network flows with high security and low latency. **Architecture**, to indicate whether the SDN controller has a centralized design, and securely connected control plane or distributed, and scalable design. **Scalability** is a fundamental feature for evaluating various SDN controllers because it is necessary to certify that the controller is capable of reducing the impact of heavy network traffic. **Reliability**, since the controller is the center of the SDN network, reliability is an essential feature for every controller since, for this feature, two significant factors are considered, elasticity and fault tolerance. The other features that have been considered for the evaluation process are **Monitoring and Visualization**, and the **Support community**. Consequently, from the various nominated features for evaluation, it can be concluded that ONOS performs better than the other controllers. The rest of this chapter is dedicated to studying the ONOS controller in detail.

Chapter 5

5 Proposed Framework (BFDD-S)

As mentioned before, the main goal of this research is to propose a near real-time framework to detect and mitigate the influence of distributed denial-of-service attacks on SDN networks. For achieving this goal, an effective method appropriate to the architecture and characteristics of the SDN network has been proposed. The proposed framework is a heterogeneous system in which the detection phase is performed out of the controller and gathering information and mitigation are performed by the SDN controller.

5.1 BFDD-S Methodology

In SDN networks, when the network traffic reaches to the SDN-based switch, the switch checks its current flow tables, If the switch finds a pre-defined policy rule for this packet in the flow tables, the flow will be passed to the destination. Otherwise, it will be considered a new flow, which is sent to the controller to make the decision. In this framework, when the new traffic flows reach the controller, it gathers all required data, aggregates them, and sends them to the detection module which resides in another server. Then, the data are evaluated by a detection module located in the attack detection server.

After analyzing the packet, if no anomaly is detected, a non-attack message will be sent to the controller. The controller then orders the switch by installing the appropriate rule in the switch's flow table to permit the related flow to pass through the switch. In opposition to that, If an anomaly is detected by the detection module, the attack detection module sends a detection message to the controller. After the controller receives the message, commands the switch to drop the flows and block the IP address of the attacker. Figure 5.1 presents the process of the network traffic received from a legitimate user and an attacker in the proposed framework in an SDN environment.

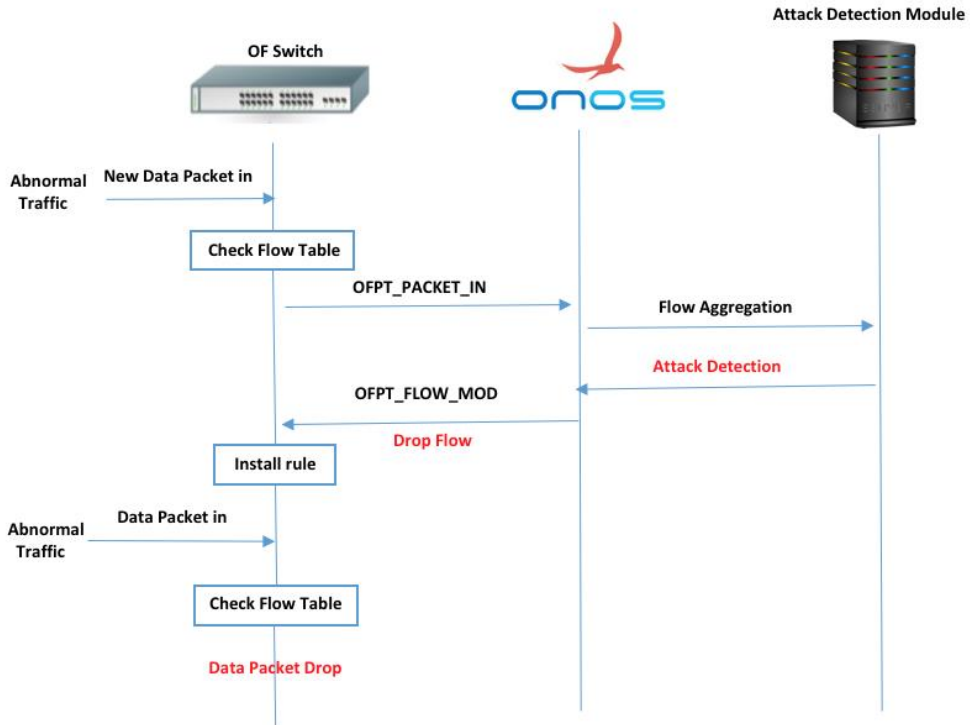
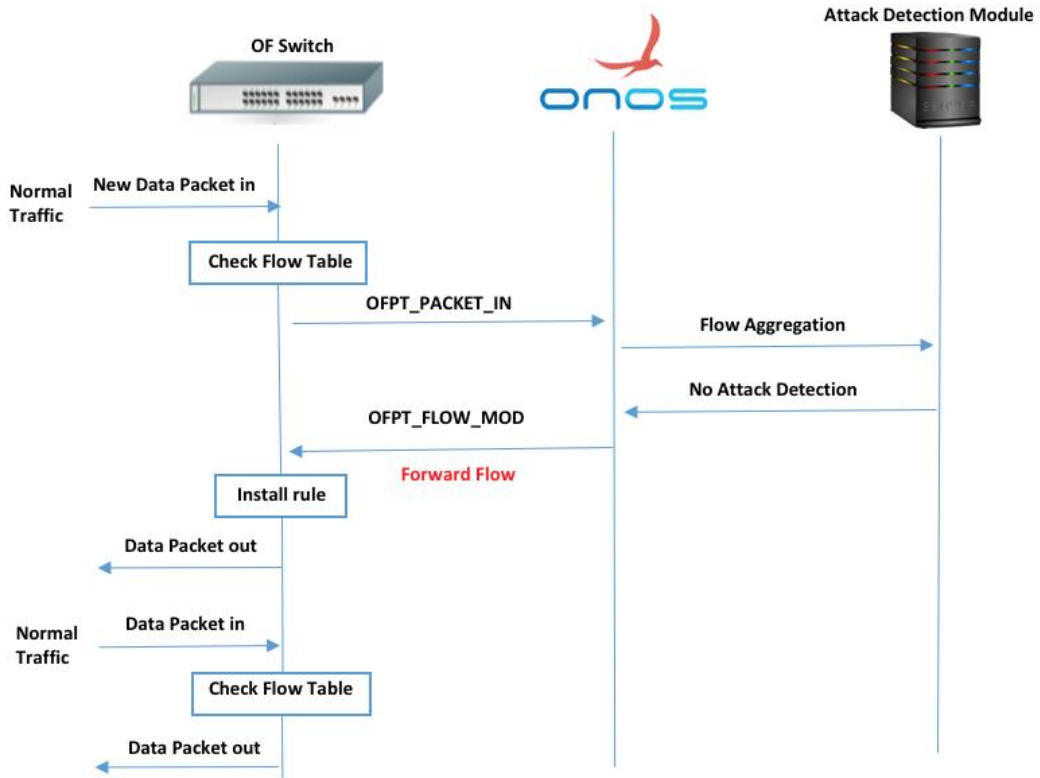


Figure 5. 1: Process of the legitimate and malicious traffic by the proposed framework

The following Pseudocode depicts the above-mentioned detection and mitigation method in the BFDD-S framework:

```

1
2 // PseudoCode for the process of the network flow in the BFDD-S framework
3
4 // Network Traffic arrive at SDN-based Switch
5 // p : Network Packets
6 // msg: Message
7 // Src: Source IP-address
8 for all p receive at OF-Switches do
9   if p is in the FlowTable then
10    forward p to Destination-Port
11  else
12    forward p to the Controller
13    // In the SDN Controller
14    if p is complies with network policies then
15      create and install Temporary FlowTable rule in OF-Switches with action:
16      FlowTable timeout = t
17      mitigate few p and forward them to Attack-Detection-Module
18      // In the Attack Deection Module
19      if any attack detects
20        generate and send Attack-Detection msg to the Controller
21      // In the Controller
22        update Log File
23        create and install a FlowTable rule in OF-Switches with action:
24        block the Src of the sender
25        drop the p
26    else
27      generate and send no-Attack-Detection msg to the Controller
28      // In the Controller
29        create and install Permanent FlowTable rule in OF-Switches with action:
30        forward all packets in the flow to Destination-Port
31  else
32    create and install FlowTable rule in OF-Switches with action:
33    drop the p
34 endfor

```

In designing this framework prototype, we utilize a combination of big data tools and machine learning together with taking advantage of the developed SDN controller, therefore we called this framework BFDD-S (Big data Framework for DDoS attack Detection in SDN network). In this section, we clarify the architecture of this framework in particular and justify how it works and depict output results, and define the functional modules for the solution that will serve as the basis for the design and development of the framework prototype.

5.1.1 Motivation behind BFDD-S

In the realm of network security, rapid attack detection is a key point for designing and developing any security system. In addition, as the architecture of the DDoS attacks shows, an attacker sends a massive volume of real or fake traffic to exhaust the network resources, with the intention to hamper or disrupt the services for normal users. Therefore, to design and implement a defense mechanism, another key point is utilizing a method that is capable of handling and processing a huge volume of data fast and in real-time.

As mentioned earlier, in designing the BFDD-S framework, we decouple the detection module from the controller to prevent controller performance degradation and delay in responding

to permissible flows sent to the controller. Therefore, in our design, we implement a big data pipeline to provide a fast and real-time data processing method using Apache Spark and prevent any delay in processing or losing any normal traffic during receiving a massive amount of traffic to the controller by using Apache Kafka.

In our detection module, we utilize Apache Kafka, as an open-source, streaming data platform, since it has capabilities such as short response time, high performance, horizontally scalable, fault tolerance, and it is capable of processing extremely fast streams of events. In the real world, it is used in real-time projects to provide data pipelines and program flow. The major advantages of using Apache Kafka in our framework include the following:

1. Creating streaming and timely data transmission lines that reliably transfer and exchange data between the controller and the detection module.
2. Creating the framework for real-time streaming data that react in time to a stream of data and transfer them.

In the detection module, we also implement Apache Spark as an open-source programming model for processing large data sets with a parallel distributed algorithm which can provide machine learning and real-time workloads. In most of the previous related research, they utilize Apache Hadoop for processing the network traffic. The main concern about MapReduce is with each step, it reads data from the cluster, performs operations, and writes the results to HDFS. Because each step requires a disk read and write, MapReduce jobs are slow due to disk I/O latency which is a big concern for detecting malicious traffic. We choose to implement Apache Spark in the data processing pipeline to overcome the above-mentioned concern in Hadoop and MapReduce.

Although Apache Spark is an open-source framework focused on machine learning and interactive queries, It does not have its own storage system. Therefore, in the data processing pipeline in the data detection module, we implement Elasticsearch for storing features that are required for data processing. Since Elasticsearch can search documents that were uploaded just one second earlier, by implementing it we can achieve a real-time search engine. In addition, It has a document-oriented distributed architecture, which makes it simple to scale up in a big environment. These real-time and scalable features are the key factors for implementing a fast, real-time, reliable, and scalable DDoS attack detection framework for SDN networks.

5.2 Framework Architecture

The aim of this work is to provide a robust and resilient intrusion detection system that can offer a fast, real-time detection system to increase the scalability and reliability of DDoS attack detection and mitigation. Figure 5.2 shows the overall view of the processing of the proposed framework.

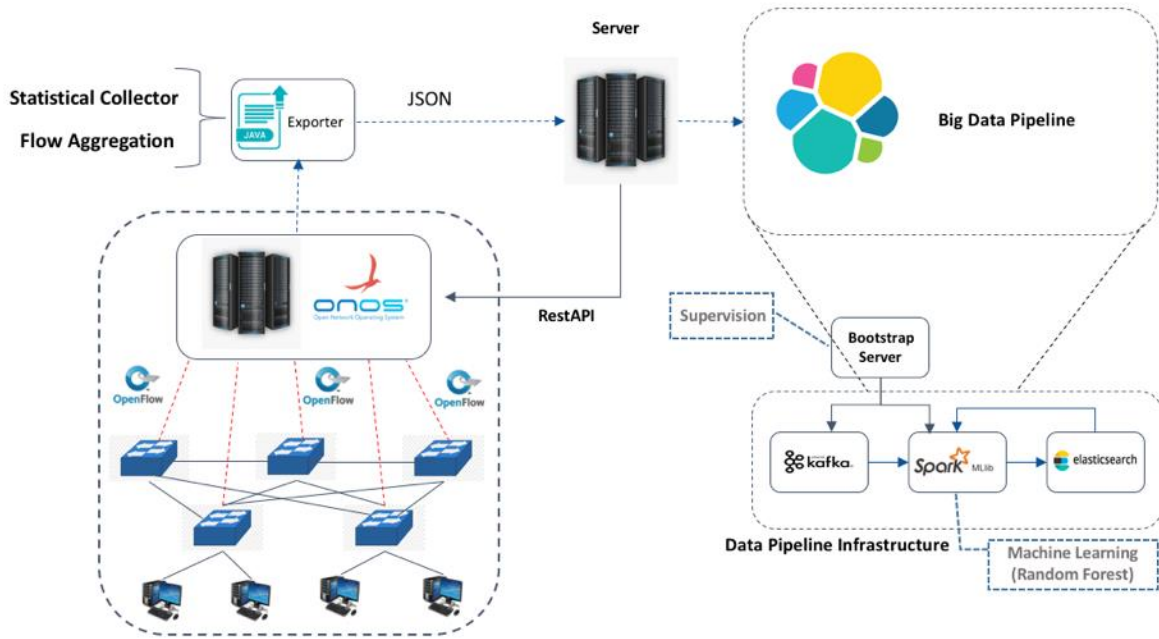


Figure 5. 2: Overall Architecture of the Proposed Framework.

The framework consists of three main modules, first module is for data gathering and formatting statistical information developed as an API to the ONOS core using Java programming language(Figure 5.3).

The major purpose of this API is to gather statistical information, aggregate it, and convert them to the appropriate format before sending it to the data processing module. In addition, the mitigation activities will be performed by the controller (We used ONOS as a controller for this experience).

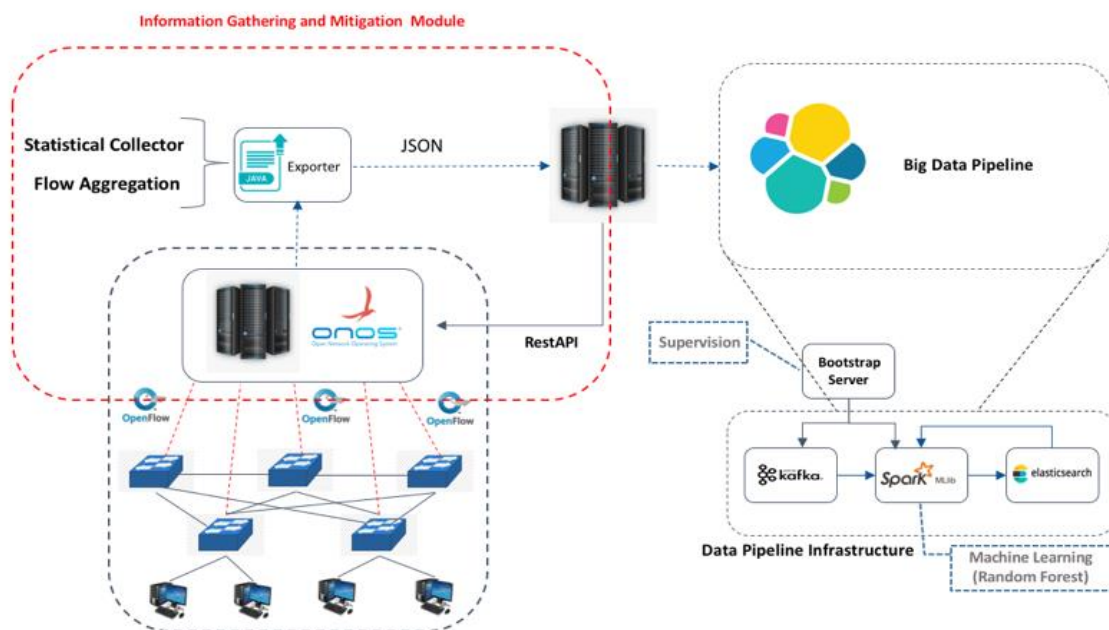


Figure 5. 3: Information Gathering and Mitigation Module.

The second module is attack detection which is located on another server and uses data pipeline infrastructure and machine learning for detecting DDoS attacks (Figure 5.4). In this framework for creating data pipeline infrastructure, we use the following big data analytic tools, Apache Kafka for message queuing, Apache Spark for data processing, and Elasticsearch for storing data.

When the new flows reach the SDN-based switches they will be sent to the controller, in this framework, the developed API module gathers, and aggregate the statistical information and then convert them to JSON format. Then the API module sends them to the detection module. In the Big Data processing pipeline, the information in JSON format first will be passed to Apache Kafka for buffering. Then Spark as a consumer reads information from Kafka's message queuing. In this module, Apache Spark is responsible for data processing and anomaly detection using the machine learning classifier.

If an anomaly is detected during the data processing phase, the attack detection module sends an incident report via the REST API to the SDN controller, providing the necessary information for decision-making, such as the source IP address of the sender.

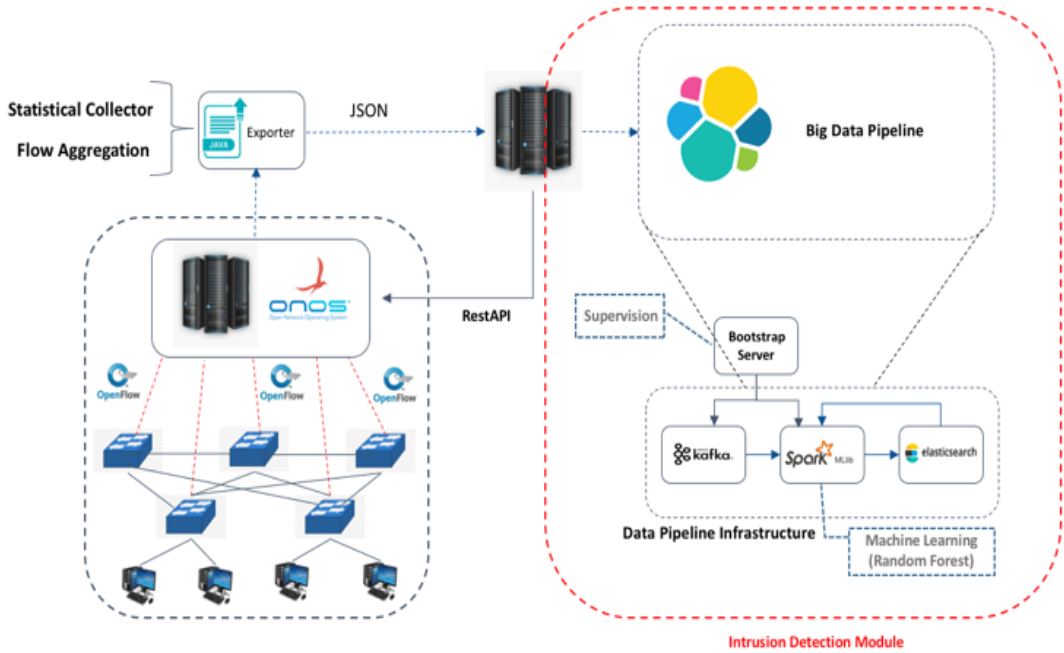


Figure 5. 4: Intrusion Detection Module.

As Figure 5.5 indicates, the third module is a machine learning model which we design and implement in the data processing phase of the big data pipeline.

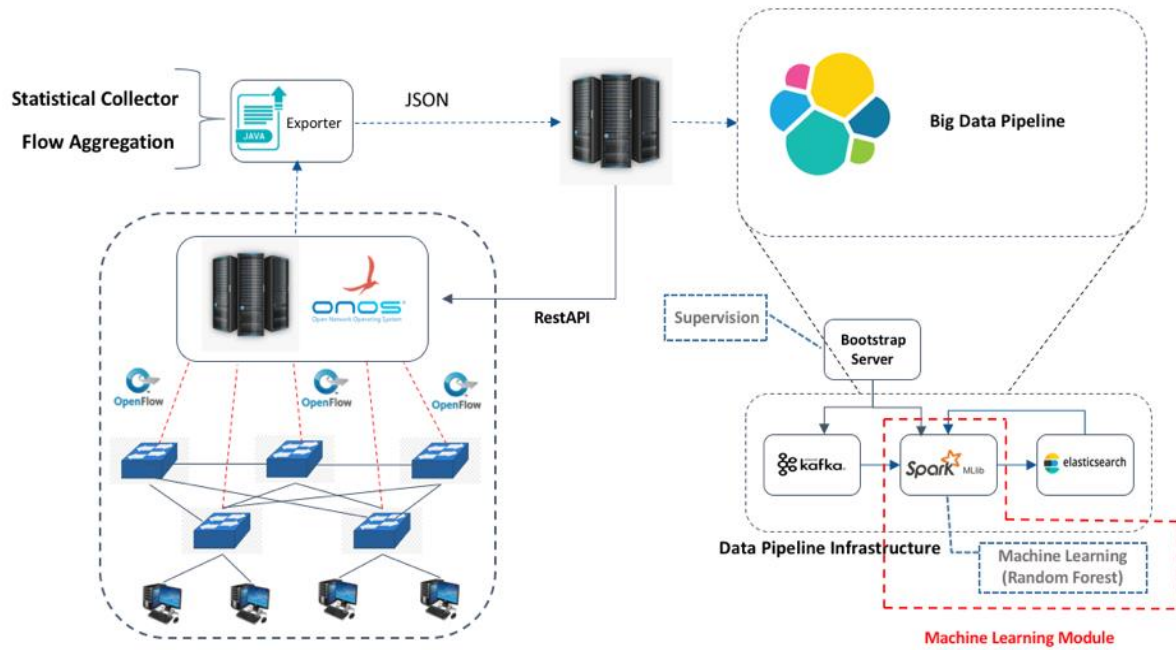


Figure 5. 5: Machine Learning Module.

In this framework, the following tools are used to create a data pipeline infrastructure: Kafka for message queuing, Apache Spark for data processing, and Elasticsearch for storing data.

In this particular case, we are going to use Kafka as the system data source for Spark Streaming. Typically, a Spark Streaming system processes a data flow, stores them in databases, and file systems, or reflects them in a graph to give a better view to the user. The operation of data processing in Spark Streaming works as follows. Spark Streaming receives the data and splits it into small batches, and these batches are processed by the Spark kernel to generate a data stream of results.

5.3 Framework Workflow

In the proposed method, every incoming packet should go through 5 different phases from the moment it reaches the switch and then be transferred to the controller and validated. Figure 5.6, demonstrates these phases in more detail.

Preliminary Phase: request for communication

The Preliminary phase is a request for communication. It is the phase when the packet for the first time enters the OpenFlow switch. Upon arrival, the switch checks its current flow tables. If the switch finds a forwarding rule defined for this packet in the flow tables, the flow is allowed to pass to the destination. Otherwise, it will be considered a new flow, which is sent to the controller to make the decision.

Phase 2: Information Gathering and Formatting

After entering the controller, initial checks such as destination address, protocol type, etc., are performed. If the packet is detected within the current network rules, the controller allows

several packets of the corresponding flow to pass to the server. The attack detection systems are usually not able to detect abnormal activities from only one packet. For this reason, the controller requests the entry of several packets of the same flows into the attack detection system. For this phase, we developed a Java API into the ONOS core, called Extractor, for gathering statistical information, flow aggregation, and converting the aggregated data to a suitable format (JSON, for instance) for sending to the data pipeline infrastructure.

Phase 3: Packet analysis and decision Making

After the second phase, it is time for the analysis and decision phase. In this phase, packets are evaluated by a detection module located in the attack detection server. The formatted aggregated flows from the Extractor module in JSON format will be passed to Apache Kafka for buffering. Then Spark as a consumer reads information from Kafka's message queuing. In this module, Apache Spark is responsible for data processing and anomaly detection. This phase consists of three steps: the feature engineering process, the machine learning process, and the decision-making step.

Spark Streaming is used as one of the components of the Apache Spark Ecosystem to enable the processing of live streams of data and to read the buffered information from Kafka. After performing Feature Engineering, the features will be stored in the index using Elasticsearch for rapid retrieval. For the machine learning process, the MLlib library is used, which is also one of the Apache Spark Ecosystem components. It is built on top of Spark core and has the facility to provide various machine-learning algorithms. For this experience, we implemented the Random Forest algorithm to detect anomaly patterns in stored data. Then it is used for detecting any anomaly in the new messages received from Kafka.

Phase 4: Reporting phase

After analyzing the packet by the attack detection module, if no anomaly is detected, a non-attack message will be sent to the controller and the controller enters the decision and action phase. The controller then commands the switch by installing the proper rule in the switch's flow table to allow the corresponding flow to pass through the switch. All these communications in both cases between the detection module and controller will be done via a REST interface. If an anomaly is detected by the detection module, the attack detection module sends an incident report via the REST API, providing the necessary information for decision-making by the controller, such as the source IP address of the sender.

Final Phase: The action phase

After the controller receives the incident report, updates its log file, and analyzes the source and destination of the flow, the controller commands the switch to drop the packet and block the IP address of the attacker.

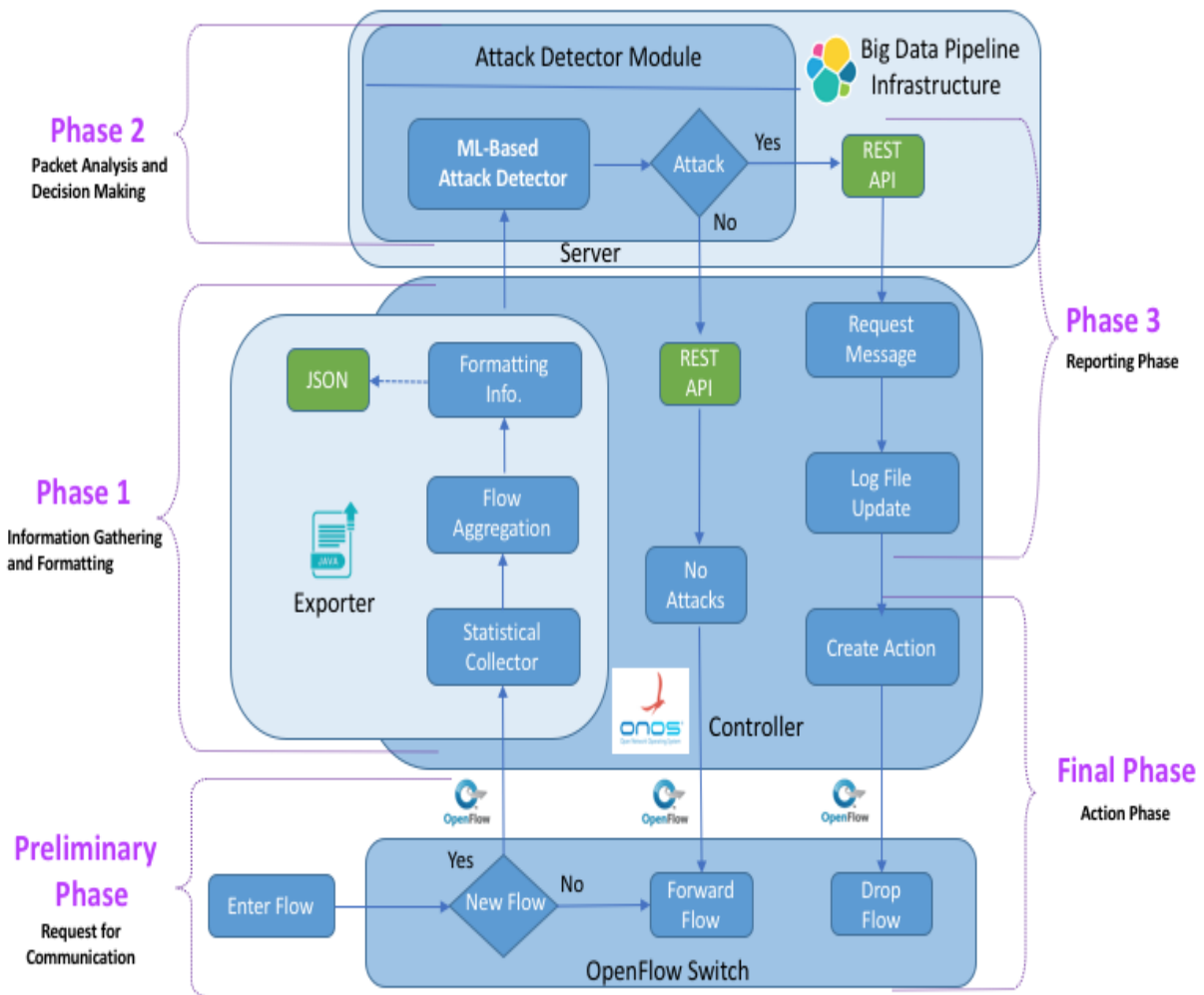


Figure 5. 6: Five Phases of the BFDD-S Framework’s packet processing.

For implementing the proposed framework, which henceforward is called BFDD-S (Big Data Framework for Detection and Mitigation of DDoS attacks in the SDN network), four phases have been accomplished as follows:

- 1- API Development for SDN controller
- 2- Algorithms Evaluation
- 3- Big Data Pipeline Implementation
- 4- Experimental Setup and Performance Evaluation

In the following sections, these four phases will be explained in detail.

5.4 API Development for SDN controller

As discussed before, ONOS implements the OSGi framework using Karaf (Figure 5.7), which breaks the application into bundles. According to the website, Apache Karaf is a minor OSGi-based runtime that offers a lightweight container onto which numerous modules and applications can be deployed. In other words, Apache Karaf is not another OSGi application, basically, its role is mostly similar to a cover on top of an already existing OSGi. On top of these runtimes, Karaf provides various value-added services. The majority of these services were

developed as OSGi bundles that are installed on containers when Apache Karaf begins. OSGi applications may benefit greatly from features such as central logging, command-line configuration manipulation, remote SSH access to OSGi runtime, and interaction with Apache Karaf's [146] flexible command shell.

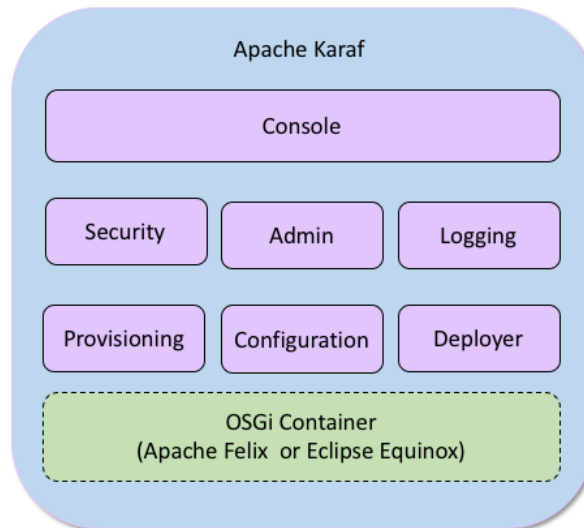


Figure 5. 7: Overview of the Apache Karaf [147].

Using these annotations provides a possibility for Karaf for calling the application's Java code during different events, such as the loading and unloading of the application's bundle. These calls are critical for initializing required variables, attaching to other services, and initiating the logic of the application. All the applications which exist in ONOS follow the Maven directory hierarchy standard. In this hierarchy standard, the application source code and libraries would be stored under the following path:

```
src/main/java
```

Therefore, in this research, for developing an API for the ONOS controller (called Exporter), The first step is to define the directory structure for this kind of bundle. This directory structure includes the pom.xml configuration file and numerous Java class files containing the application's main functionality and for keeping the source code of tests, the following path has been used:

```
src/test/java
```

Component Templates can be used to automate the creation of a skeleton directory structure for any typical project. It is necessary for all the applications created and developed for ONOS to be linked to Karaf with the use of annotations like as `@Activate` or `@Deactivate`.

Afterward, by using these annotations, write entry and exit functions to enable Karaf to successfully load your application. To register additional services that are accessible with Karaf, the `CoreService.registerApplication()` routine of the `org.onosproject.core` should be used inside the startup method. The `coreService` package is used to register the application with a unique name so that Karaf can identify it.

Karaf requests all the services that the application requires to use via the startup and cleanup processes. As soon as the skeleton is complete and ready to be bonded using Karaf, it is time to create the application core logic, often known as the business logic. The core logic is developed using Java files, which are subsequently compiled into the necessary class files. When the writing code is finished, the Maven commands are used to build/compile the application.

Apache Maven is a project management software and download tool. Based on the concept of the Project Object Model (POM) in the project, it can manage the construction of a project. This management starts from the time of compilation and continues until publication and documentation. This method allows teamwork to take place properly. The major objectives of the Maven are exposed in the following figure.

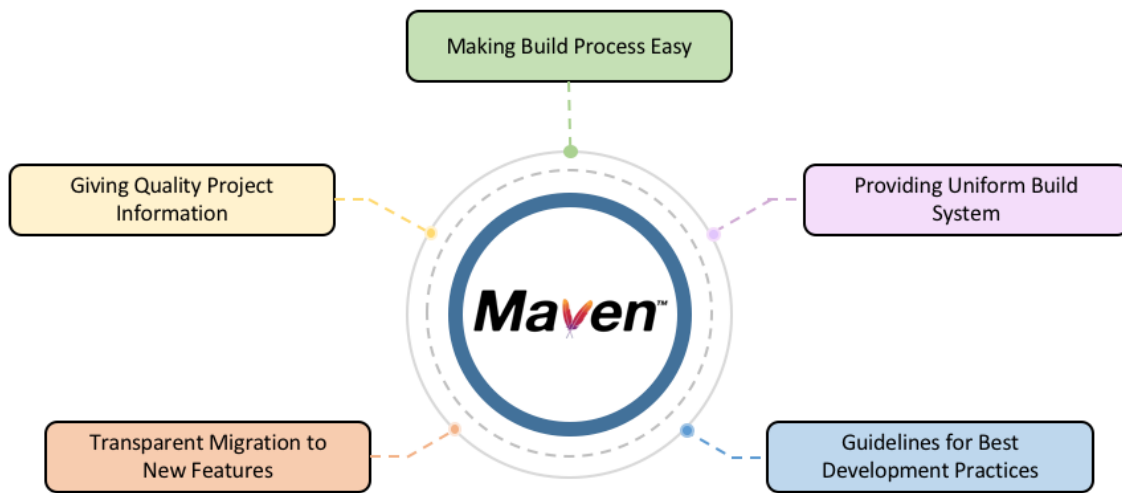


Figure 5. 8: Maven Objectives [146]

It is one of the most widely used Java tools for packaging, building, deploying, compiling, testing, and documenting projects. It allows all developers to follow a standard for generating a project, instead of using their own taste. And in the future, the other developers can easily communicate and continue developing the project structure.

Maven is responsible for compiling all the Java files of the application, connecting all of the dependent bundles together, or downloading those dependency bundles that are not currently accessible locally.

Once the process of Maven is complete, the result which is the Java archive (jar) would be stored in a position inside the ONOS folder where Karaf would be able to access it without the need for further developer participation. To complete the installation process, launch the application using the ONOS CLI with the "feature: install application name>" command. For a better understanding of the essential steps for developing an API into the ONOS controller, the aforementioned steps are shown in Figure 5.9.

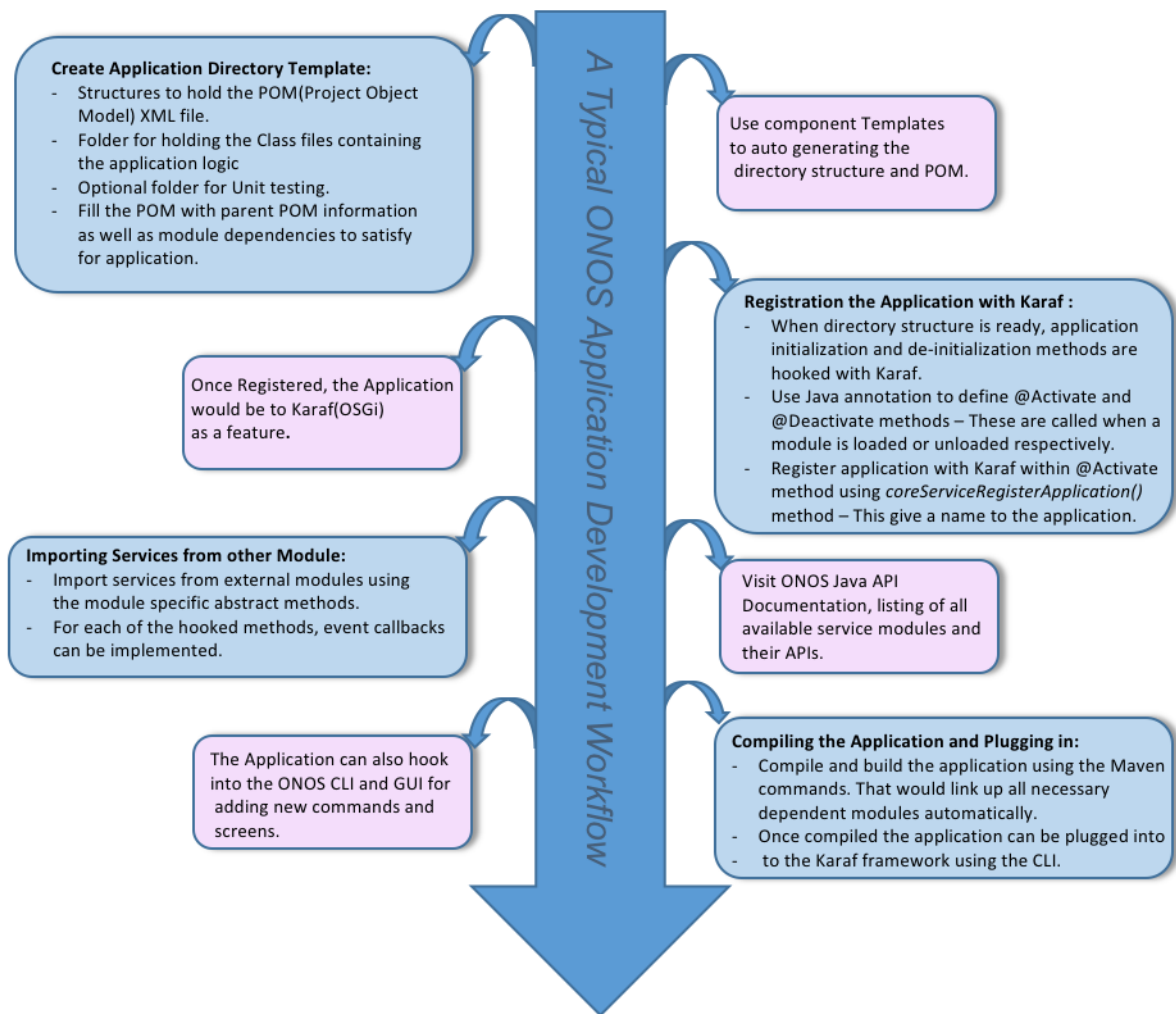


Figure 5. 9: A typical development process for an ONOS application [148].

ONOS also adds two more folders to this structure, one in the following path for keeping the application's source code:

Src/main/java/org/onosproject/Exporter

And the other one is for keeping the source code of the test in the following path:

Src/test/java/org/onosproject/Exporter

There must be a BUILD file outside of this structure, at the same level as the src directory, that informs Bazel what to build and how to compile it, defining the artifacts and their dependencies. As mentioned earlier, for creating an ONOS app, two methods have a significant role, more precisely, in at least one of its source code files, every ONOS application must implement an Activate and a Deactivate method. As the name implies, these methods specify what should be done when the program is activated and what should be done when it is deactivated. Apart from these, an ONOS application can be developed by creating as many methods as required.

To start, the applications folder, which is located in the ONOS root directory, is used to create a new application considering the structure described in the previous section. Following the writing of the code and creating the file in which the source code for the program will be stored, it is necessary to create a skeleton of the application. For creating an application for ONOS the following services are needed:

Core Service, Statistic Service, Device Service, and Metrics Service.

5.5 Machine Learning Algorithms Evaluation

The machine learning method is the most human answer to complex and demanding activities. It is one of the specializations in the area of artificial intelligence and a data mining technique. is a technique in which a computer is trained to perform a specific task using specific algorithms. For the preparation phase, collected data is used called data set to train the machine to perform certain operations by collecting data called data sets. This training continues to guarantee that the machine is capable of doing the same task as a human with similar quality. This process is also called learning with the observer because the machine can continue to function due to the training given to it by humans.

5.5.1 Algorithm Methods

According to different methods, various models can be defined, and among all these designed models, the best one should be selected based on the evaluation method. Analysis methods provide required information and other statistical data used for evaluating different models. Model processing is often called training and is a procedure for applying a specific mathematical algorithm to the data of a structure and extracting patterns. The type of patterns found in the training process depends on various items such as the selection of training data, the selected algorithm, and how the algorithm is implemented.

In the world of machine learning the algorithm can be classified into three main categories:

- Supervised Machine Learning
- Unsupervised Machine Learning
- Reinforcement Learning

5.5.1.1 Supervised Learning

In this method, the data should be labeled. Most machine learning methods use supervised learning. In supervised machine learning, the system tries to learn from the prior examples provided. In other words, in this type of learning, the system tries to learn the patterns based on the examples given to it. Mathematically speaking, when the input variable (X) and output variable (Y) are available, a mathematical algorithm can be used to derive an input-to-output mapping function based on these variables, this is called supervised learning. The mapping function is represented by $Y = f(X)$.

This type of algorithm can be divided into two categories: classification and regression methods. Classification: A problem can be considered as classification when the output

variable is a category or group. An example of this would be a sample belonging to the "black" or "white" categories and an email to the "spam" or "non-spam" categories. Regression: A problem is considered as regression when the type of the output variable is real for instance "height". In other words, the classification algorithms work with discrete variables and regression algorithms work with continuous variables.

5.5.1.2 Unsupervised Learning

In unsupervised learning, the algorithm alone must look for appropriate patterns in the data. From a mathematical point of view, unsupervised learning is used when there are only input variables (X) in the data set and no associated output data variables. This type of learning is called unsupervised because, unlike supervised learning, there is no correct answer given, and the machine itself must look for the answer. In other words, when the algorithm uses a dataset that lacks labeled data (output variables) to work, it uses another mechanism for learning and decision making. In this way, learning is done on unlabeled data, and the system itself must discover hidden patterns in the data, using different tasks such as clustering, association, and dimensionality reduction. In this case, the model learns through observations and discovers the relations and structures in the data set. When a data set is introduced to the model, the model automatically discovers the connections and patterns in the data using clustering.

5.5.1.3 Reinforcement Learning

Reinforcement learning is a type of machine learning approach that designs algorithms for recognizing the environment and making optimal decisions to achieve the maximum set of rewards. This approach, not only has a rich mathematical structure and leads to the emergence of profound theories and robust learning methods, but also it is very flexible and has been widely used in practical problem-solving. The reinforcement learning problem is formulated mathematically based on Markov decision processes.

In this method, the algorithm enters the cycle of trial and error, learns to make certain decisions, and thus is constantly learning. Reinforcement learning also refers to the ability to find a feature related to the external environment to achieve the best results. The concept is also known as the trial and error model. This factor is scored or penalized based on the right or wrong results, and finally, the model achieves the ability to improve through the positive scores and the desired results. This learning and improvement will continue until the system can make the necessary predictions and decisions about the new input data. Figure 5.10 depicts the three above-mentioned classified categories and correlated prevalent algorithms.

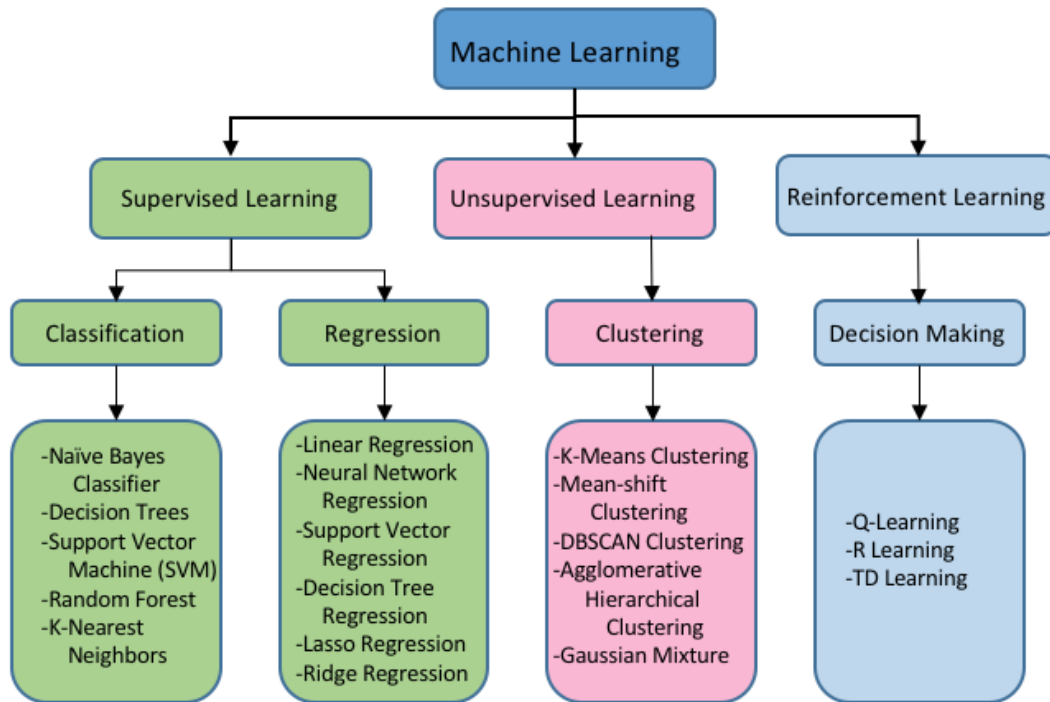


Figure 5. 10: Various Types of Machine Learning Algorithms

5.5.2 Process of Designing a Model in this Research

One of the most important components for success in projects is the use of accurate methodology and workflows for project management. The following are the necessary steps to implement analytical models based on . Through this research, various sources have been studied and they have introduced the stages of differ in terms of the number of stages. Most sources mainly introduced five, six, or seven steps. Of course, this is not a fundamental difference, but in some sources, it is preferred that some stages be separated and not more detailed or not. Therefore, it is possible that even in some sources we encounter more than 7 steps, so here we examine a 10-step process for as shown in Figure 5.11.

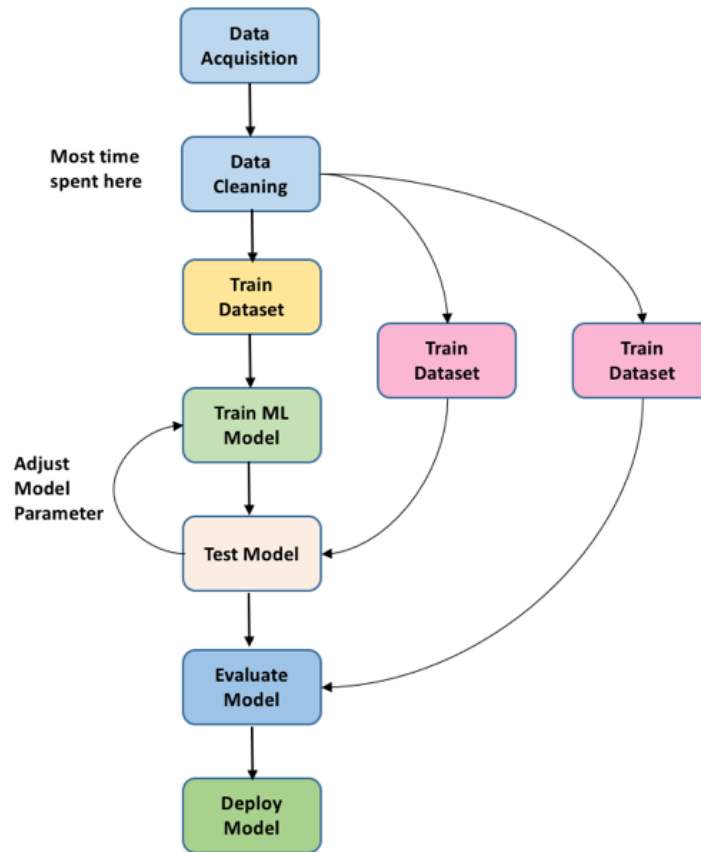


Figure 5. 11: Process of Design Model [149]

These ten steps in the process are repeated periodically to improve the performance of the model until it reaches the desired quality.

5.5.3 Evaluate the Accuracy of a Model

Sensitivity and specificity are two essential indicators for assessing statistically the functioning of the classification model. The data can be distributed into positive and negative groups, and the sensitivity and specificity indicators may be used to quantify and characterize the performance of an experiment when categorizing data into these two groups.

5.5.3.1 Confusion Matrix

In the field of artificial intelligence, a confusion matrix is a matrix in which the performance of relevant algorithms is represented. The name of this matrix is also derived from the fact that it was easier to see the error and the interference between the results. This matrix shows the results of the classification based on the actual information available. Then, Different criteria for classification assessment and accuracy measurement might be created based on these results (Figure 5.12 indicates all the evaluation indicators which can be derived from Confusion matrix).

There are different metrics that can be considered when determining the suitability of a classification algorithm. On the one hand, we must consider the efficiency of the algorithm when classifying the analyzed elements. To measure it we will use the following metrics:

True Positive (TP): represents the number of malicious packages correctly classified as such.

True Negative (TN): represents the number of legitimate packets correctly classified as such.

False Negative (FN): represents the number of malicious packets erroneously classified as legitimate.

False Positive (FP): represents the number of legitimate packets erroneously classified as malicious.

Once a model is designed, one of the most important phases is to evaluate its performance. In the following, the methods for evaluating a model are briefly defined:

		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive(TP)	False Negative(FN) Error Type II	Sensitivity $\frac{TP}{(TP + FN)}$
	Negative	False Positive(FP) Error Type I	True Negative(TN)	Specificity $\frac{TN}{(TN + FP)}$
		Precision $\frac{TP}{(TP + FP)}$	Negative Predictive Value $\frac{TN}{(TN + FN)}$	Accuracy $\frac{TP+TN}{(TP + TN + FP + FN)}$

Figure 5. 12: Overall View of the Confusion Matrix and Assessment Methods [149].

5.5.3.2 Selection of Metrics

Finally, the obtained results are evaluated and interpreted. The following criteria are usually considered in the evaluation, as shown in Figure 5.13.

Accuracy: The most prevalent, fundamental, and straightforward metric for evaluating a model's value is its accuracy. This parameter indicates the number of patterns that have been correctly identified and is formulated and defined based on the confusion matrix. In general, accuracy refers to the model's ability to properly anticipate output. Nevertheless, this criterion provides only a limited amount of information regarding the model's performance.

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

Precision: It determines, how true is the result when the model predicts the result positively. When the value of false positives is high, the correctness criterion will be appropriate. In fact, when the researcher uses this parameter as an evaluation parameter for his category, the goal is to achieve the utmost accuracy in identifying positive class samples. The precision is calculated as follows:

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

Recall: The accuracy of negative class identification, in contrast to this characteristic, may sometimes be significant. The specificity parameter, often known as the "True Negative Rate," is one of the most frequently used characteristics that is typically taken into account together with sensitivity. This parameter is calculated as follows. The ratio of the correct number of items classified by the algorithm from one class to the number of items in that class is calculated as follows:

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

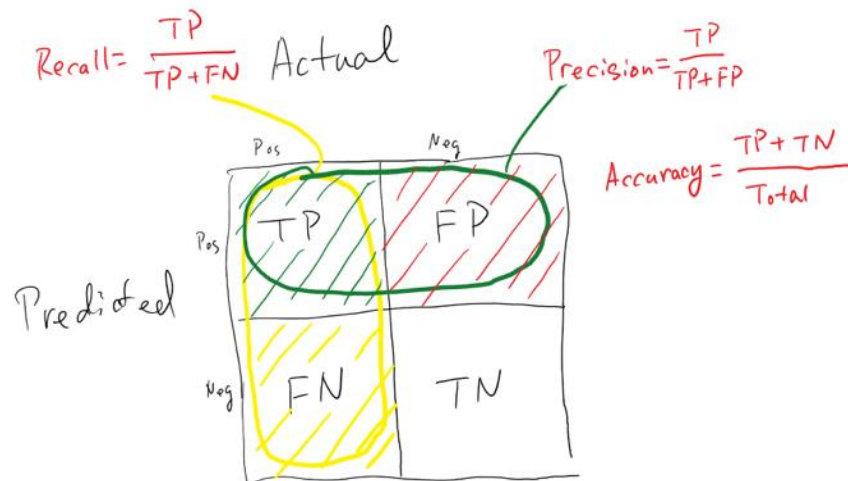


Figure 5. 13: Three main assessment Metrics [150].

5.5.3.3 Evaluation of the Model in a Real Environment

After analyzing the data and creating insight, it is time to present and distribute this output. Showing the results to competent people at the right time and using modern methods provides the conditions that can be used to make a valuable reservoir of insight before

deciding to do anything. If adequately filled and properly removed, a repository can improve performance in an organization. Today, different methods are used to present data. In this regard, several tools have been designed and implemented that can be used to provide and distribute insights.

5.5.4 Classification Algorithms

For this research, the five most well-known classification algorithms have been chosen and performed evaluation process, to select one of them to be implemented in the data processing module of the proposed framework for DDoS attack detection. In this section, these five classification algorithms will be introduced.

5.5.4.1 Logistic Regression (LR)

Logistic Regression is a method and is one of the most popular techniques for classifying data. In the classification problem, it is used when one class must be distinguished from another. This algorithm is used to predict a class-dependent variable using a given set of independent variables. Logistic Regression is one of the most widely used algorithms in the field of . This technique is a supervised learning method, and the data have a specific label, and the learning process is done to classify based on this data and their labels [151].

Table 5.1 describes pseudocode for classification with the LR algorithm.

Table 5. 1: Pseudo code for the Logistic Regression algorithm [152].

```

-----
Algorithm Logistic Regression
-----
given  $\alpha, \{(x^i, y^i)\}$ 

initialize  $a = \langle 1, \dots, 1 \rangle^T$ 

Perform feature scaling on the examples' attributes repeat until Convergence.

for each  $j=0, \dots, n$ :

 $a_j = a_j + \alpha \sum_i (y^i - h_a(x^i)) x_j^i$ 

for each  $j=0, \dots, n$ :

 $a_j = a_j$ 

output  $a$ 
-----

```

5.5.4.2 K-Nearest Neighbor (KNN)

The k-Nearest Neighbors (k-NN) algorithm is a supervised classification algorithm based on neighborhood criteria. In particular, the concept behind k-NN is that fresh samples will be categorized with the same class that has the largest number of most similar neighbors to them

in the training set. It is based on a hyperplane computation. Neighbor K method is used for many methods because it is effective, non-parametric, and easy to implement. However, the categorizing process takes a long time, and it is difficult to determine the best value for k. In general, if the best choice of K is given to the data, the influence of noise is decreased, and the boundary between classes is less distinct [151]. Table 5.2 explains pseudocode for classification with the KNN algorithm.

Table 5. 2: Pseudocode for the KNN Algorithm [152].

```

-----
Algorithm k-Nearest Neighbor (KNN)
-----
start

Let  $S = \{a_1, a_2, \dots, a_n\}$ , where S signifies the training set and  $a$  signifies
article documents.

 $k \leftarrow$  the desired number of nearest neighbors

Compute  $d(x, y)$  between new instance  $i$  and all  $a \in S$ 

Select the  $k$  closest training example to  $i$ 

 $Class_i \leftarrow$  best voted class

end
-----

```

5.5.4.3 Naïve Bayes Classifier (NB)

This algorithm can be called a generative model. This implies that it is supposed that the input data can be demonstrated by a model and its certain parameters, and the learning phase is comprised of attempting to determine which values are appropriate. This is due to the fact that distinguishing the distribution followed by the input data is often more difficult than creating an appropriate classifier without doing so, compare to other classification techniques. This algorithm simplifies the Optimal Bayes Classifier based on the Bayes theorem which refers to a considerable decrease in the number of parameters that must be anticipated by an assumption. In a particular label, it is assumed that the characteristics of the input vector are independent of one another. This assumption is made with the awareness of the fact that it is often inaccurate, especially in intrusion detection cases, but that it significantly simplifies the construction of a probabilistic model. That is the reason this classifier is referred to as Naïve, which means the maximum possibility principle is considered to estimate the parameters. Therefore, it will be mandatory to identify the ideal parameters for these algorithms to make them work properly [151].

Table 5.3 depicts pseudocode for classification with the Naïve Bayes algorithm.

Table 5. 3: Pseudo code for the Naïve Bayes Algorithm [152].

Algorithm Naïve Bayes

start

Let $S = \{a_1, a_2, \dots, a_n\}$, where S = training set and a = articles:Calculate the probability of the classes $P(C)$ Calculate the likelihood of attribute A for each class $P(A|C)$ Calculate the conditional probability $P(C|A)$

Assign the class with the highest probability

end

5.5.4.4 Decision Tree (DT):

The Decision Tree is one of the most widely used techniques in data mining. Decision trees are one of the simplest algorithms available, but they are also one of the most effective in the field of . This technique can be beneficial when the volume of data is very high. It is a collection of algorithms that categorize data and then make a sequence of choices based on that classification and a tree structure is used to describe this series of choices [153].

In general, data categorization is carried out from the root node to an appropriate leaf node, where each leaf node represents a different category [154].

Throughout the supervised training phase, this method generates a binary decision tree, which is can be described as if-then rules. The decision is made on each node depending on the value of an attribute associated with that node. During the training phase, the choices and the sequence in which they are made are decided based on their relevance to the classification of the data. In this algorithm for a particular dataset S with N classes where $k \in \{1, 2, 3, \dots, N\}$. Variable p_k signifies the number of cases classified as k inside the dataset. The summation of the probabilities of the incorrect classification of k is mentioned as [155]:

$$p_w = \sum_{r \neq k} p_r = 1 - p_k$$

Table 5.4 illustrates the procedure of the classification with the decision tree algorithm.

Table 5. 4: Pseudo code for the Decision Tree algorithm [152].

Algorithm Decision Tree

start

\forall attributes a_1, a_2, \dots, a_n

Find the attribute that best divides the training data using information gain

$a_{best} \leftarrow$ the attribute with the highest information gain

Create a decision node that splits on a_{best}

Recurs on the sub-lists obtained by splitting on a_{best} and add those nodes as children of a node

end

In general, this algorithm demonstrates a function that accepts a vector of attributes as input, and as a result, it returns a particular value that shows the "decision". There are two types of decision trees; Classification trees are models in which the objective variable has distinct values, whereas regression trees are models in which the target variable has continuous values.

5.5.4.5 Random Forest (RF):

The random forest method is an example of an ensemble algorithm for classification using a large number of individual decision trees. As a hybrid classification system, this technique employs a mixture of two or more classification algorithms, which are also referred to as fundamental algorithms. The decision tree method is the foundation of the Random Forest algorithm. The technique shown in Table 5.5 is used to create individual trees. As it was mentioned earlier in this chapter, the combination of a number of different models in an ensemble method can offer an improved level of accuracy [151].

Table 5. 5: Pseudo code for the random forest algorithm [152].

Algorithm Random Forest

Require IDT (a decision tree inducer), T (the number of iterations), S (the training set), μ (the subsample size), N (the number of attributes used in each node)

start

$t \leftarrow 1$

repeat

$S_t \leftarrow$ Sample μ instances from S with replacement.

Build classifier M_t using $IDT(N)$ on S_t

$t++$

until $t > T$

end

5.5.5 Intrusion Detection Using Techniques

Using techniques for intrusion detection first came to the sight of researchers in 1987 [156]. Since then, it's been a hot study topic around attack detection for researchers. A well-tuned machine learning system may be capable of detecting not just known attack patterns but also novel ones. As machine learning technologies have progressed over time, numerous algorithms for intrusion detection have been developed and employed. In the scope of intrusion detection, Decision trees, neural networks, Bayesian algorithms, support vector machines, and genetic algorithms have all been successful techniques.

Additionally, unsupervised techniques have been used to identify intrusions by using ensemble methods, graph theory, and clustering. Machine learning has a lot of ability for intrusion detection since it particularly excels at recognizing patterns in huge amounts of data. In most situations, there is just too much data to process for data analysts and scientists. They must go through massive log files for signs of malicious behavior. This is the area in which different machine learning techniques are particularly good at [157].

5.5.6 Data Set Selection

For researchers, finding a suitable collection of data to train machine learning models for intrusion detection is one of the most challenging concerns in this subject.

Datasets are a set of information with specific properties that are collected in a specific area. Datasets or datasets are collections of data that are used with the same subject matter for analysis and data mining projects. Of course, there is another application of datasets for

comparison between different methods, in that, for example, on a data set, two different methods (algorithms) are implemented and according to the results can be based on the criteria of accuracy, speed, and complexity of each Compared methods.

Because of the amount of data passing through the network and the need for confidentiality, most data sets used in supervised learning comprise simulated data. In the Third International Competition for Knowledge Discovery and Data Mining, the first dataset which is called KDD Cup 1999 was generated [158]. Developing a network intrusion detection system was the main objective of the competition. The dataset was created using data collected over nine weeks from a simulated Air Force local area network. Then, a CSV data set was created based on the process of the collected raw data from the network. This data set consists of the sample data for different types of the attacks such as DoS attacks, illegal approaches to a local private account, illegal access to the network from a remote computer, and Probing attacks.

However, even though the KDD Cup 1999 data set was convenient for the original competition and following research, for more than 15 years, it was nearly the only data set utilized for intrusion detection investigation. Since, in the last 15 years, computers and networks have evolved tremendously, the KDD 1999 data set was complained about including duplicate information and representing old-fashioned technology. Because of the above-mentioned weaknesses of the KDD 1999, numerous scientists have generated more up-to-date data sets to solve the KDD Cup 1999's drawbacks. A more recent network data set was presented by the UNSW-NB15 data set, which was produced in 2015 [159]. In this section, these three well-known datasets will be briefly introduced but according to the above-mentioned issues about KDD CUP 1999, the two other data sets have been used in this research to perform the evaluation of various machine learning algorithms for selecting the appropriate algorithm for intrusion detection to implement in the proposed framework.

5.5.6.1 KDD CUP 1999:

It is based on the DARPA 1998/1999 dataset, which the MIT Lincoln Lab generated from an emulated network environment. In this data set, both regular traffic and various forms of computer attacks such as Dos, buffer overflow, etc. were logged for several weeks. As indicated earlier, KDD CUP 99 is a stream-based dataset that is developed by data from system logs and network packets engaged in the connection. in addition to typical traffic, and has already been separated into two subsets: a training subset for machine learning algorithms and a test subset. Conversely, the Canadian Institute of Cybersecurity (CIC), the organization that is presently providing this dataset, also has additional up-to-date datasets with comparable features that are accessible for free to use [158].

5.5.6.2 NSL KDD Data Set

As an upgrade to KDD CUP 99, it was developed in 2009, and it has addressed some of the redundancy issues mentioned in [160]; however, it still retains some of the issues raised by McHugh in [161]. Despite some issues mentioned above, it is still regarded as a strong benchmark for evaluating various intrusion detection techniques, and it remains to be one of the best choices for researchers for their investigations in the field of intrusion detection [162]. Comparable to KDD CUP 99, NSL-KDD also uses stream-based data that is developed with information about the computers and packets involved in connections. In this section,

the structure and content of the NSL-KDD are described briefly, according to the information provided by the CIC website and analysis performed by [163]. NSL-KDD dataset is an established benchmark for evaluating network intrusion detection techniques. It is produced from the KDDCUP99 dataset, which has a significant disadvantage of a large quantity of unusable duplicate records existing in it. This may produce a wrong result and prediction specifically for machine learning algorithms as a detection algorithm. Therefore, NSLKDD eliminates repeated records from the KDDCUP99. Specifically, NSLKDD includes two training sets ('KDDTrain' and 'KDDTrain 20percent') and two test sets ('KDDTest+' and 'KDDTest-21'), among expressing 41 features defining the statistical information and fundamental aspect of the network [164].

The dataset contains a total of 41 characteristics which can be classified into the following four comprehensive categories: R2L attack, U2R attack, DoS attack, and PROBE attack. In this data set, each item has a label that corresponds to the flow grouping. In addition, in this data set, the flows are categorized into two types, either normal or conforming as an attack, and contain the data equivalent to 40 individual kinds of attacks. Table 5.6 presents the different features of the NSL-KDD data set. For this research, the features numbers F2, F5, F6, and F23 are used because these features can be simply achieved from the SDN controller; therefore, they can be appropriate options for this study [165].

Table 5. 6: List of features of the NSL-KDD dataset.

F#	Feature name	F#	Feature name	F#	Feature name
F1	Duration	F15	Su attempted	F29	Same srv rate
F2	Protocol type	F16	Num root	F30	Diff srv rate
F3	Service	F17	Num file creations	F31	Srv diff host rate
F4	Flag	F18	Num shells	F32	Dst host count
F5	Source bytes	F19	Num access files	F33	Dst host srv count
F6	Destination bytes	F20	Num outbound cmds	F34	Dst host same srv rate
F7	Land	F21	Is host login	F35	Dst host diff srv rate
F8	Wrong fragment	F22	Is guest login	F36	Dst host same src port rate
F9	Urgent	F23	Count	F37	Dst host srv diff host rate
F10	Hot	F24	Srv count	F38	Dst host serror rate
F11	Number failed logins	F25	Serror rate	F39	Dst host srv serror rate
F12	Logged in	F26	Srv serror rate	F40	Dst host rerror rate
F13	Num compromised	F27	Rerror rate	F41	Dst host srv rerror rate
F14	Root shell	F28	Srv rerror rate	F42	Class label

Table 5.7 shows the distribution of the flows stored in the NSL-KDD data set for the identified and new attacks based on four different attack classifications [166].

Table 5. 7: Divisions of famous and new attacks in the KDD-Test set.

	DoS	R2L	U2R	Probe
Known attacks	5741	2199	37	1106
	76.98%	79.85%	18.50%	45.68%
New attacks	1717	555	163	1315
	23.02%	20.15%	81.50%	54.32%

5.5.6.3 UNSW-NB15 Data Set

KDDCUP99 and NSLKDD benchmark data sets were created ten years ago to evaluate research efforts on network intrusion detection systems. Conversely, numerous recent studies have demonstrated that in the context of the current network security environment, these data sets do not accurately represent network traffic and contemporary low footprint attacks [161]. The UNSW-NB15 dataset was generated in 2015 at the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) using the AXIA Perfect Storm tool to provide a combination of the realistic modern normal and malicious network traffic. The genuine current normal and the contemporary synthetic attack activities of the network traffic are both included in this data set to make it a hybrid data set. To construct the characteristics of the UNSWNB15 data set, both existing and innovative methodologies are applied. [159].

In this data set, by using three virtual servers, the data was collected by an IXIA traffic generator. Two servers were set up to generate legitimate network traffic, while a third was set up to produce malicious network traffic. Argus and Bro-IDS software retrieved 49 features from the raw network packets, including flow-based and packet-based characteristics, and nine distinct attacks including, Shellcode, Backdoors, Reconnaissance, Exploits, Analysis, Generic, DoS, Fuzzers, and Worms [167]. Packet-based features are extracted from the packet header and its payload (also called packet data). On the other hand, flow-based characteristics are produced by exploiting the sequencing of packets as they move across a network, starting at a source and ending at a destination. This dataset is included 2,540,044 realistic modern normal and abnormal (also known as an attack) network activities. As Figure 5.8 shows, the data set is divided into two parts, with 175,341 records in the training data set and 82,332 records in the testing data set, respectively [168]. Compared to previous benchmark datasets like DARPA98(LABORATORY, 1998), KDDCUP 99, and NSL-KDD, the structure of this dataset is more complicated. As a result, the UNSW-NB15 is enhanced to provide a more comprehensive and reliable assessment of the current network intrusion detection technologies [159].

Table 5. 8: Number of records in training and testing subsets for each class

Classes	Training Subset	Testing Subset
Normal	56,000	37,000
Analysis	2,000	677
Backdoor	1,746	583
DoS	12,264	4,089
Exploits	33,393	11,132
Fuzzers	18,184	6,062
Generic	40,000	18,871
Reconnaissance	10,491	3,496
Shellcode	1,133	378
Worms	130	44
Total Number of Records	175,341	82,332

5.6 Selection of Proper Machine Learning Algorithm

In this phase, to select appropriate machine learning algorithms that should be implemented in the data processing step in the attack detection module, five prevalent classification algorithms have been evaluated by using two popular datasets. In this research, two popular data sets have been used, and evaluate different classification algorithms using the python library Scikit-learn; therefore, the above-mentioned machine learning process is customized into the following steps depicted in Figure 5.14.

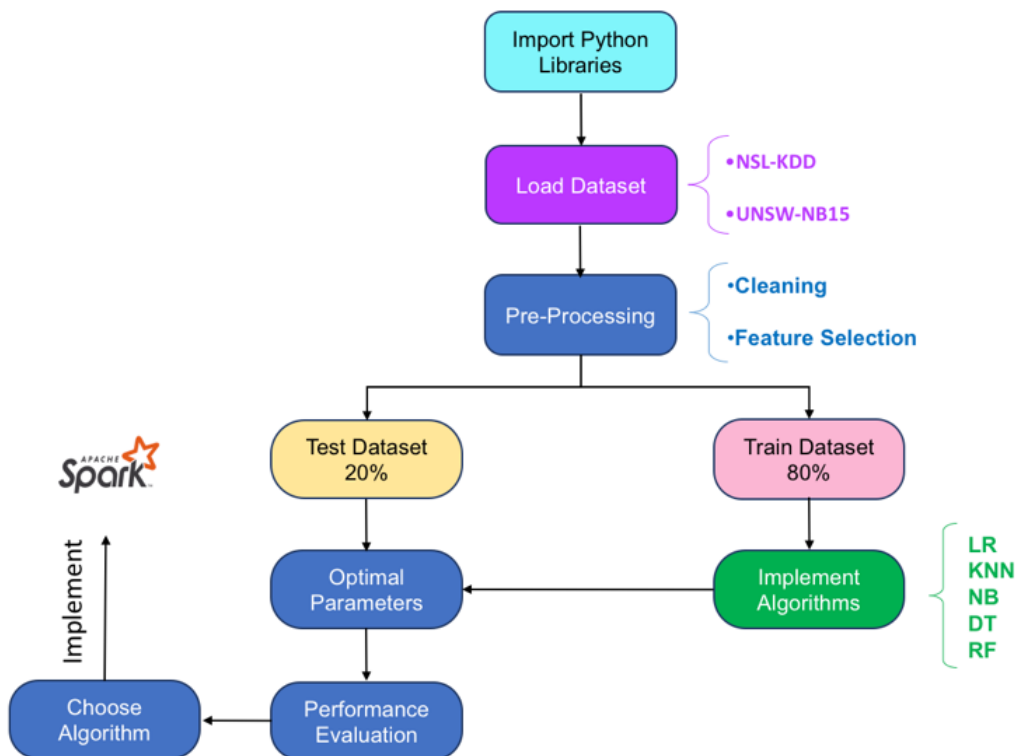


Figure 5. 14: The Machine Learning workflow used in this research.

The main reason behind using two data sets for the evaluation phase of this research is, that there is some research [161] showing that the NSL-KDD data set, despite the fact that it is the most frequent data set used by researchers for training and test designed machine learning model for DDoS attack detection, it contains the data that has not been collected from a real environment and most of the data captured in the simulation network. In addition, this data set was created in 2009, therefore it does not contain data related to novel and modern attack methods and it is getting slightly out of date.

For implementing and evaluating different ML algorithms the following python libraries have been used: NumPy, pandas, Scikit-learn, and matplotlib, and can be seen in figure 5.15.

For this reason, a Python script is implemented so that, each of the algorithms, applies the ML algorithm to the dataset. First to train the model and later to test its effectiveness. The introduction of a validation phase between the two will be studied. This script will be based on the Scikit-Learn and matplotlib libraries.

5.6.1 Data Preprocessing

In the world of machine learning, data processing is so vital. It means that before implementing any machine learning algorithms, data must be prepared to increase the accuracy and output of the work. As shown in figure 5.14 in this research, for data preprocessing, the following steps have been performed:

To perform data preprocessing, the initial step is to import the Python libraries, which in this study Scikit-learn, Pandas, NumPy, and Matplotlib have been used (Figure 5.15). These libraries are very beneficial for entering data and managing them.

```
#import libraries

import os
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt; plt.rcParams()
import matplotlib.pyplot as plt
# Applying k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
from IPython.display import display
```

Figure 5. 15: Importing the essential Python Libraries.

As revealed above, in this research two popular data sets have been used. First, the NSL-KDD data set has been downloaded and with the help of Pandas library can work with the data format. The relevant commands are shown in the figure 5.16.

```
#importing dataset
datafile_handler = open("/home/gwdguser/Downloads/source/second/KDDTrain+.csv", "r")
# creating a Pandas DataFrame using read_csv function
# load data from a csv file.
dataset = pd.read_csv(datafile_handler, sep = ",")
# closing the datafile handler
datafile_handler.close()
```

Figure 5. 16: Importing the Data set and Creating the Pandas Data Frame.

In pre-processing phase, the figure 5.17 present the steps we follow to prepare data set for the train and test the designated machine learning algorithms.

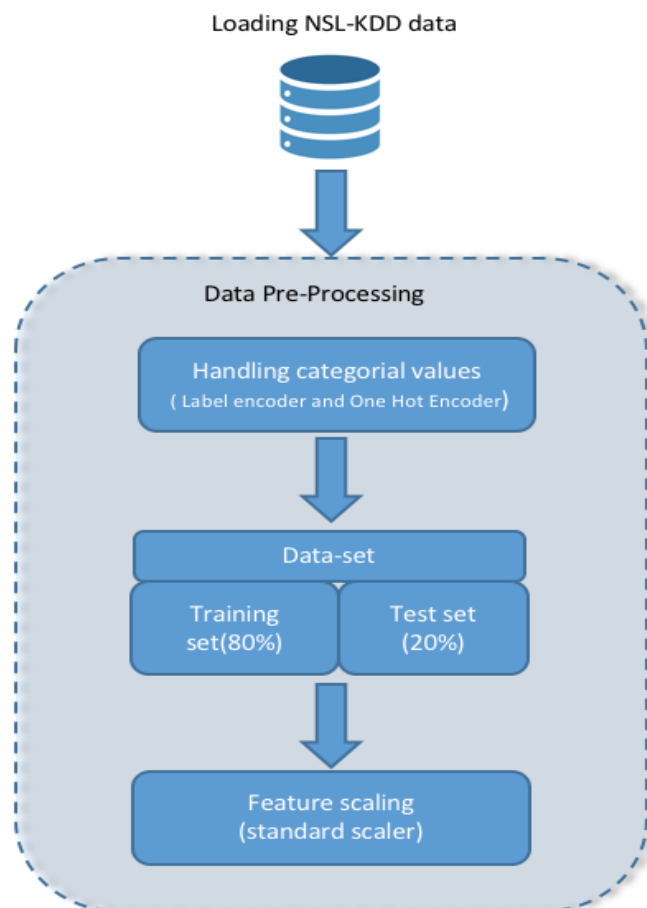


Figure 5. 17: Data pre-processing workflow

The next step is to convert qualitative data. Any data that is not numerical is qualitative or categorical. For modeling, the data must be numerical. To do this, import the "LabelEncoder" class from the "sklearn.preprocessing" library and create a labelencoder_X object from the LabelEncoder class. We then use the fit_transform method to convert the data as displayed in figure 5.18.

```
#encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

#first applying label encoding to convert strings to number.
labelencoder_x_1 = LabelEncoder()
labelencoder_x_2 = LabelEncoder()
labelencoder_x_3 = LabelEncoder()
x[:, 1] = labelencoder_x_1.fit_transform(x[:, 1])
x[:, 2] = labelencoder_x_2.fit_transform(x[:, 2])
x[:, 3] = labelencoder_x_3.fit_transform(x[:, 3])
```

Figure 5. 18: Convert Qualitative Data

After converting the values, we must make changes to the numbers again. Suppose we attribute the number 1 to red and the number 2 to green. In this case, machine learning algorithms consider the number 2 to be superior to the number 1. If we did not have such an intention. We use One-Hot Encoding so that the numbers are not superior to each other in terms of algorithm.

One-Hot Encoding is used where numbers are not hierarchical. These are just numbers that have no superiority over others. For One-Hot Encoding, we use the following method (See figure 5.19):

```
# Now applying one-hot-encoding to map the numbers to appropriate weights.
onehotencoder_1 = OneHotEncoder(categorical_features = [1])
x = onehotencoder_1.fit_transform(x).toarray()
onehotencoder_2 = OneHotEncoder(categorical_features = [4])
x = onehotencoder_2.fit_transform(x).toarray()
onehotencoder_3 = OneHotEncoder(categorical_features = [74])
x = onehotencoder_3.fit_transform(x).toarray()
```

Figure 5. 19: Applying One-Hot Encoding

In this stage of preprocessing with Python, we divide our data into two sets, one to train our designed model, entitled the training set, and the other to test the effectiveness of our model. The division is generally 70/30. To do this, we enter "train_test_split" from the "sklearn.model_selection" library.

```
from sklearn.model_selection import train_test_split
```

We will now create 4 sets to build our training and experimental collections:

X_train: Examples of training part

X_test: Examples of the test section

Y_train: Tutorials tags

Y_test: Test labels

The `test_train_split` function finds samples and labels along with the training-to-test ratio and breaks it down into 4 variables (Figure 5.20)

```
#splitting dataset into train and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Figure 5. 20: Splitting Dataset

Most machine learning algorithms use Euclidean distance for their calculations. For this reason, if a few samples have a very high or very low value, the modeling accuracy decreases. Data scaling is used to solve this problem. One of the most famous of these comparisons is the Z conversion. The Z conversion is done using the "StandardScaler" class in the `sklearn.preprocessing` library as shown in figure 5.21.

```
#Feature Scaling of the NSL-KDD dataset.
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
x_train = sc_x.fit_transform(x_train)
x_test = sc_x.transform(x_test)

df2 = pd.DataFrame(x_test)
display(df2.head())
display(x_train)
```

Figure 5. 21: Data Scaling

This was the last stage of data preprocessing. Now you can inject your data into machine learning algorithms. For the UNSW-NB15 data set similar steps for the machine learning process are done and in the next section the accomplished results from these two data sets compare to each other to select the most appropriate classification algorithm to be implemented in the data processing pipeline.

5.6.2 Comparison of the Classification Algorithms

As indicated earlier in this section, in this research, for the DDoS detection phase, we assessed the effectiveness of several classification algorithms in order to identify the most appropriate classifier to deploy using the Spark Mllib library (Figure 5.22). The mission of the detection phase is to classify whether the incoming network traffic is normal or consider it an attack, based on the features of the incoming traffic.

To evaluate the efficiency of the different Spark machine learning-based algorithms, we compared various classic machine learning algorithms, such as Logistic Regression, Naïve Bayes, K-Nearest Neighbor, Decision Tree, and Random Forest using two datasets i.e., NSL-KDD and UNSW- NB15 as well-known datasets for Network Intrusion Detection Systems.

```

# Training the Selected Algorithms
# Create the RandomForest Model
RFclassifier = RandomForestClassifier(n_estimators=200, bootstrap = True, max_features = 'sqrt')
# Create the Decision Tree Model
DTclassifier = DecisionTreeClassifier(criterion = "entropy", max_depth=10)
# Create the Naive Bayes Model
NBclassifier = GaussianNB(priors=None, var_smoothing=1e-09)
# Create the Logistic Regression Model
LRclassifier = LogisticRegression(solver = 'lbfgs', max_iter=100)
# Create the KNeighbors Model
KNNclassifier = KNeighborsClassifier(n_neighbors=5)

```

Figure 5. 22: Training Algorithms

Table 5.9 indicates the classification report and result for the supervised algorithms we have used in this research for the NSL-KDD data set. In this evaluation for simplifying, we use an abbreviation for these classification machine learning algorithms as follows:

LR: Linear Regression, KNN: K-Nearest Neighbor, NB: Naïve Bayes, DT: Decision Tree, and RF: Random Forest.

Table 5. 9: Accuracy Comparison Table (Using NSL-KDD data set)

ML Algorithms	NSL-KDD Dataset		
	Accuracy	Precision	Recall
LR	0.89525	0.89513	0.89541
KNN	0.97352	0.97543	0.97421
NB	0.88152	0.88131	0.88123
DT	0.98591	0.98585	0.98593
RF	0.99705	0.99711	0.99701

The result presents Decision Tree and Random Forest gives the best accuracy which is more than 98%, followed by the Logistic Regression algorithm which gives more than 89% accuracy. This classification report depicts that the Naïve Bayes has the weakest performance, with a score of 88% accuracy.

Although the two classifiers delivered more than 98% accuracy, the table indicates that Random Forest has the best result and Decision Tree comes in second.

The results for the UNSW-NB15 dataset are shown in table 5.10.

Table 5. 10: Comparison Table (Using UNSW-NB15 data set)

ML Algorithms	UNSW-NB15 Dataset		
	Accuracy	Precision	Recall
LR	0.96316	0.93408	0.93532
KNN	0.96139	0.96421	0.96308
NB	0.88171	0.77398	0.63133
DT	0.98157	0.98967	0.97131
RF	0.99283	0.99259	0.98358

Obviously, the Random Forest algorithm is significantly better than the other classification algorithms followed by the Decision Tree, K-Nearest Neighbor, Logistic Regression, and Naïve Bayes algorithms with an accuracy of 99%, 98%, 96%, 96%, and 88% respectively.

Additionally, an interesting observation in the above classification report is that the Naïve Bayes algorithm achieved high scores (>88%) in accuracy; however, the algorithm also produced low scores (>64%) in recall and precision (>78%) which indicates that here again, using UNSW-NB15 data set the Naïve Bayes has the weakest performance compared to the other classification algorithms we use in this research (See table 5.10).

Furthermore, to give a better view of the comparison between different machine learning algorithms with two different data sets, the following figures have been used. Figure 5.23 indicates that the large majority of machine learning models achieved 95% or better scores through each different evaluation metric, including accuracy, precision, and recall.

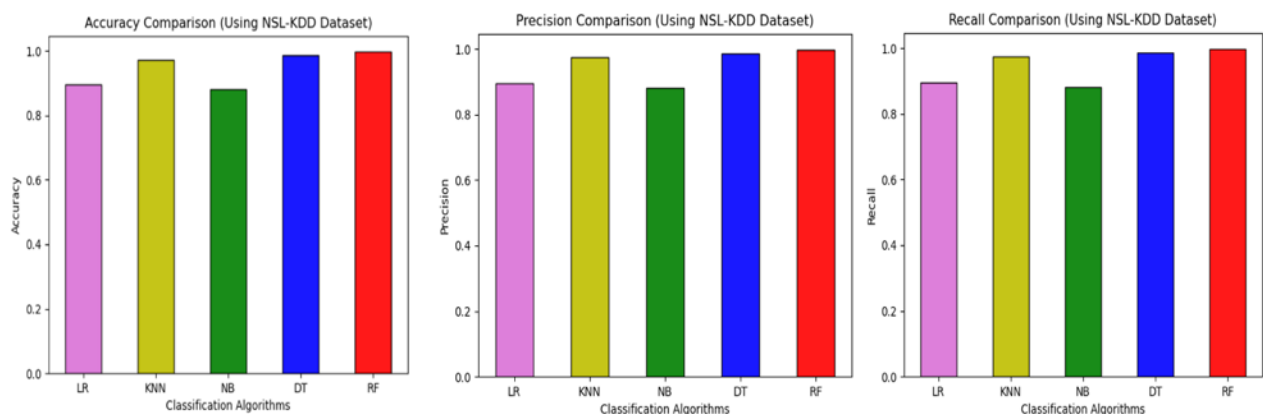


Figure 5. 23: Comparison of different ML Algorithms using the NSL-KDD dataset.

The only algorithm that produced a noticeably low score in all three evaluation metrics is the Naïve Bayes algorithm. The accuracy of the Naïve Bayes algorithm depicts a score below the 88% threshold in both data sets we have used in this research (Figure 5.24).

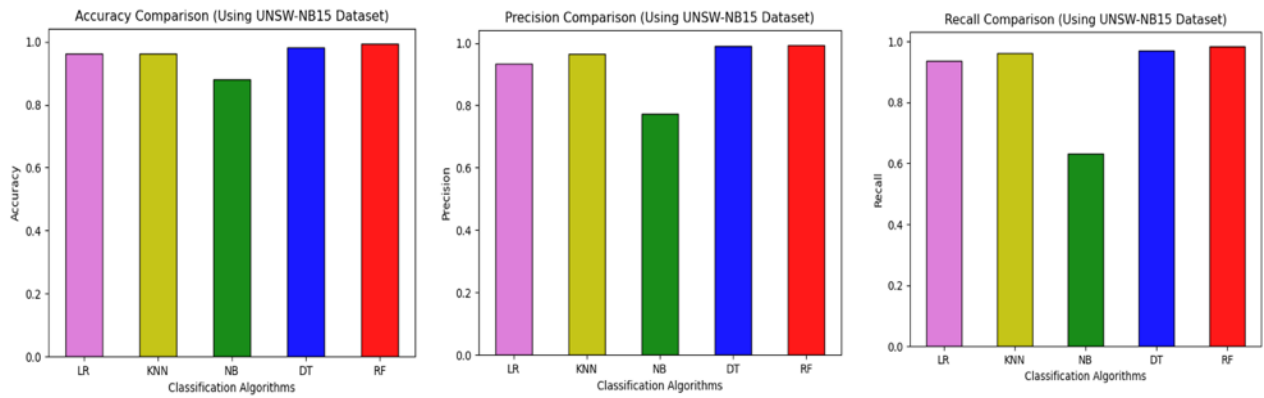


Figure 5. 24: Comparison of different ML Algorithms using the UNSW-NB15 dataset.

For better overall observation, figure 5.25 presents the results of the evaluation of these classification algorithms using both data sets in one graph for each one of the three considered metrics.

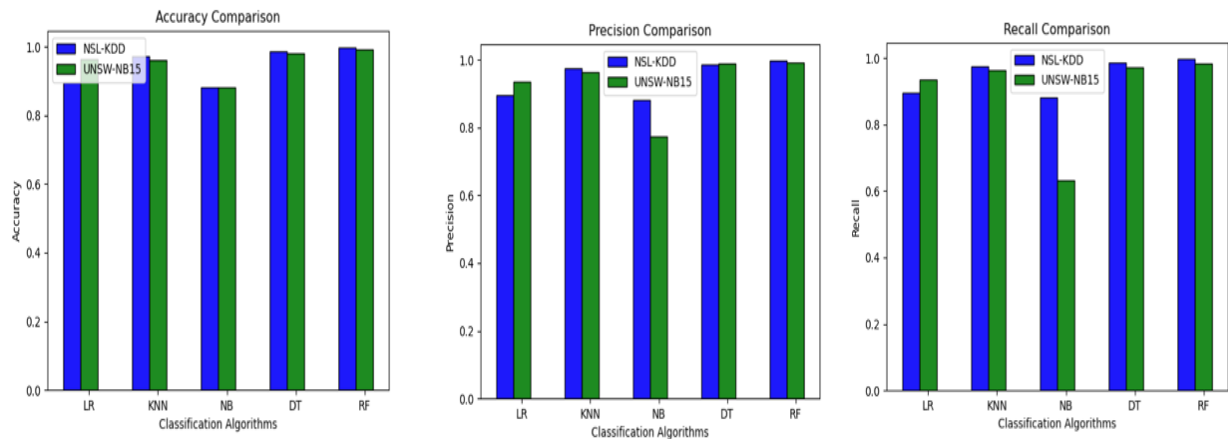


Figure 5. 25: Comparison of NSL-KDD and UNSW-NB15 datasets results for different ML Algorithms.

Based on the evaluation results, we concluded that the Random Forest classifier offers the best performance in terms of accuracy, precision, and recall compared to the other classification algorithms. Therefore, in this research, we choose this algorithm to be implemented in the data processing phase of our proposed data pipeline using Apache Spark for attack detection.

5.7 Tools Used for Proposed Framework (BFDD-S)

In this research, several tools and applications are used to provide an appropriate platform for implementing our proposed framework. we divide the tools into three categories, General Tools, big data Tools, and the SDN controller. Therefore, in this section, we introduce all the tools that we use in this research.

5.7.1 General Tools

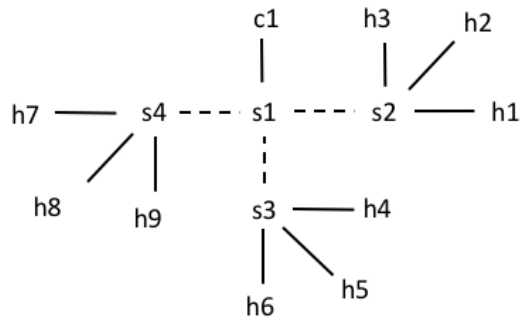
To evaluate the performance of our proposed framework, it is required to provide a platform to emulate an SDN network using SDN-based network equipment, then perform a DDoS attack against the SDN controller, and monitor the critical system metrics. To achieve this objective we use the following tools:

5.7.1.1 Mininet

Mininet is a network emulator which means that a virtual network comprising hosts, switches, controllers, and connections can be built using Mininet. which creates a network of virtual hosts, switches, controllers, and links. It is a software environment that most research teams, students, and network researchers use to model, emulate, analyze, and evaluate the performance of SDN (software-based networks) networks and their associated protocols. Using the Mininet emulator, an SDN network can be modeled before the actual implementation, and various network parameters can be changed during the emulation, and the results can be evaluated and compared. The various modules in this simulator were developed to support different types of controllers and switches. In addition, complex custom scenarios can also be created and experimented with using the Mininet extensible API library in the Python programming language [169].

Standard Linux networking software is used by Mininet hosts, and OpenFlow is supported on its switches for high flexibility in traditional and SDN routing. For supporting all the research that requires to have a comprehensive tentative network on the PC or laptop, Mininet can provide a learning, testing, developing, debugging, and prototyping environment. Among the above-mentioned major features of Mininet, Other capabilities which Mininet has can be defined as follows [169]:

- For emerging OpenFlow applications, it offers a simple and reasonable network test bed.
- It provides a possibility that various developers can work simultaneously and individually on an identical topology.
- Without any requirement to utilize a real physical network, a complex network topology can be defined and tested using Mininet.
- For launching tests and debugging throughout the network, it contains a CLI that maintains topology and OpenFlow. Figure 5.26 presents a sample CLI command. This command will create a binary tree SDN network topology of a specified depth with 9 hosts, connected to 4 switches (depth=2 and fanout -3) and a local SDN controller. Switches will interact with the controller using vSwitch and OpenFlow version 1.3. The result of the executed command will be the following topology:



```

gwdguser@gwdguser-machine: ~
File Edit View Search Terminal Help
gwdguser@gwdguser-machine:~$ sudo mn --topo tree,depth=2,fanout=3 --switch ovsk,of13 --
controller=remote,ip=127.0.0.1 --mac
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s2, h1) (s2, h2) (s2, h3) (s3, h4) (s3, h5) (s3, h6) (s4, h
7) (s4, h8) (s4, h9)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet>

```

Figure 5. 26: Sample Mininet CLI command.

5.7.1.2 Open vSwitch

Open vSwitch is open-source software designed to perform as a virtual switch in a virtual machine. It is responsible for switching network traffic between virtual machines located on the same physical computer, as well as communication between virtual machines and the physical network. Moreover, it is compatible with various protocols, for instance, OpenFlow and tunneling protocols such as IPsec or GRE [170].

Similar to SDN-based physical switches in the SDN environment, Open vSwitch can operate under the supervision of an SDN controller; in this case, it performs as a learning-based Ethernet switch. It has the capability to be configured through commands conducted by the command line console (CLI). As figure 5.27 illustrates an OVS switch consists of three modules, two modules operate in user space and one in the kernel space [171]:

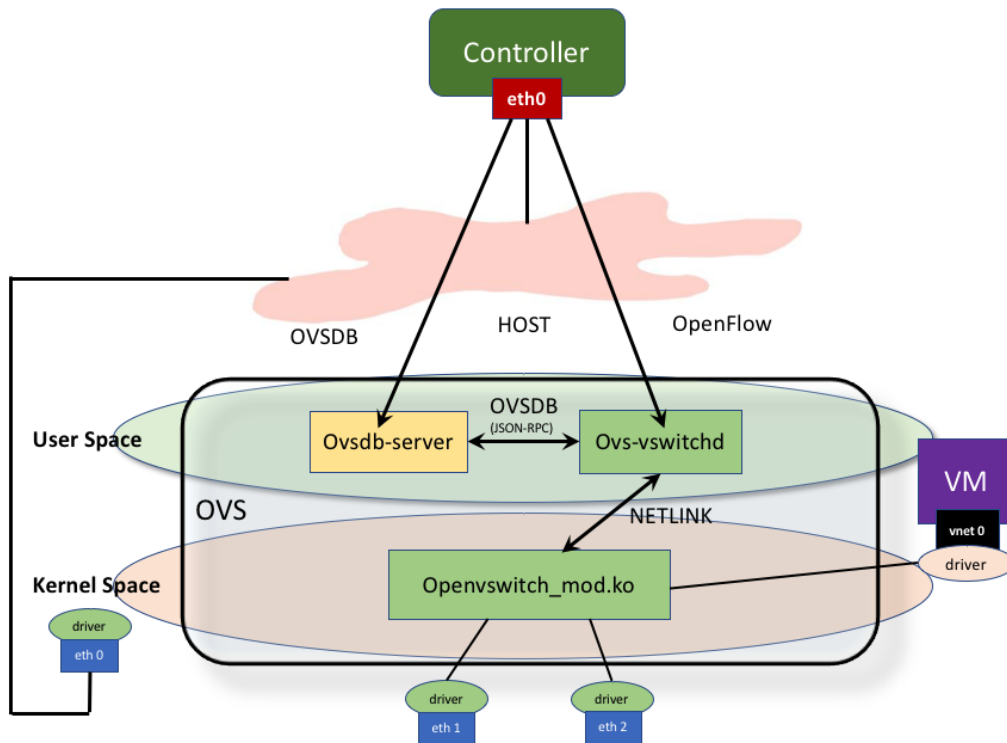


Figure 5. 27: Open vSwitch Architecture [172].

- **Ovsdb-server:** This module operates in user space, and it is the location where switch configuration is stored. It utilizes the OVSDB protocol for communicating with the ovs-vswitchd and with a controller.
- **Ovs-vswitchd:** this module operates in user space, and it is responsible to receive packets from a new flow in the network and deciding in what way they should be treated. If there is a controller, it will communicate with it using the OpenFlow protocol.
- **Openvswitch_mod.ko:** this module operates in the kernel space; its role is to route all the packets belonging to a flow that has already matched in the flow table of the switch. In other words, when a packet arrives at the switch, it goes straight to the kernel. If there is no match for that flow in the flow table, it is routed to Ovs-vswitchd, which is in charge of routing it and creating a new entry in the flow table. These two modules communicate with each other through the NETLINK protocol. The benefit of designing and developing such a module is that the operation runs in the kernel at a much higher speed.

5.7.1.3 Hping3

The hping3 is a command-line tool for producing and analyzing network packets. hping3 is developed and supported by antirez@invece.org and is licensed under GPL version 2.

It has the power to create and send custom TCP/IP packets and present the received replies from the target system. In addition, the Hping3 tool can produce custom network packages that can be used for TCP / IP security testing, such as port scanning, firewall rules assessment,

and network performance analysis. This tool has the ability to simply send echo request data in various formats with ICMP protocol and it also supports other prevalent protocols such as TCP, UDP, and even raw IP protocol. This package can simply be used in network scanning processes to achieve more accurate information about the target system [173].

5.7.1.4 Htop

Htop is a command-line utility that is an enhanced version of the top command. It displays critical system metrics such as running tasks, PID, uptime, load average, memory utilization, CPU consumption, and many other vital statistics information. What makes this tool stand out from its other predecessor tools is the capability to demonstrate system metrics in a more organized way. This allows users to intuitively determine the system metrics they need with simplicity compared to the other utility tool [174].

5.7.1.5 Programming Libraries

Scikit-Learn¹² (also known as Sklearn), is a free Python library for implementing machine learning algorithms. This library contains the main ML algorithms, as well as all required tools for pre-and post-processing. These tools include utilities for different purposes, such as dividing datasets and for extracting and analyzing data resulting from a test. Scikit-learn is selected over other similar libraries such as Java WEKA for its versatility and comprehensive accessible documentation.

Numpy¹³ is the main library for mathematical calculations in the Python language. This library provides facilities for defining and managing arrays for computing projects. Before Numpy, a similar version called Numeric was created by Jim Hugunin with the participation of several other developers. In 2005, Travis Oliphant created Numpy by keeping Numarray's features (As a more flexible alternative to Numeric) in Numeric and making extensive changes.

Pandas¹⁴ is one of the open libraries created to work with data with a relational (rational) or labeled structure. This library provides a variety of data structures along with the possibility of applying numerical operations on these data and has the capability to work with time series. Pandas is based on the NumPy library, and many NumPy structures are used and extended in this library.

The advantages of this library include the following:

- High speed and efficiency in working with data.
- Ability to load data from different source files.
- Transform data flexibly.
- Providing the possibility to work with time series.
- The possibility of grouping data according to practical purposes.

¹² <https://scikit-learn.org/stable/>

¹³ <https://numpy.org/>

¹⁴ <https://pandas.pydata.org/>

Matplotlib¹⁵ is a free Python library developed for data plotting that we use to plot and visualize the evaluation results.

5.7.2 Big Data Analytics Tools

In the realm of cybersecurity in general, and network security specifically, it is crucial to transform the data into useful information for the security expert or application. For this purpose, big data analytic tools can provide valuable facilities for processing, analyzing, and storing all the collected data. In this research, we also utilize some big data tools to provide a data processing pipeline to detect malicious traffic. In this section we describe these tools:

5.7.2.1 Apache Spark

Spark [175] is one of the quickest-growing and most widely accepted big data tools today. It represents a great opportunity for organizations to gain the benefits of large-scale data analytics. Apache Spark has lately emerged to participate in large-scale data analysis. This fast-processing engine was produced in 2009 at Berkeley University. And then, as its name suggests, it has been developed within the framework of the Apache project, which guarantees its Open-Source license. Apache Spark is designed and developed as a distributed processing engine that is responsible for orchestrating, distributing, and monitoring applications that consist of multiple data processing tasks by several worker machines, which form a cluster. Spark's architecture has a modular design that allows you to adopt different configurations according to need.

In addition to the flexibility provided to Spark by its programming design, the master-slave computing model provides it with scalability and fault tolerance, Apache Spark is a framework that allows us to process distributed data quickly and efficiently, capable of orchestrating, distributing, and monitoring applications by reading data from different storage systems on several clustered machines. Although Spark reads data from different storage solutions, it does not store data in itself but is focused on processing very quickly and efficiently, since all this processing is done in memory, which drives this technology to be one of the best solutions for real-time data processing (Streaming). Spark is responsible for the management and coordination of tasks with data on a cluster of computers. This cluster of computers is orchestrated by a cluster manager, such as Spark Standalone Cluster Manager, Apache YARN, or Apache MESOS. The applications (Spark Applications) are sent to the cluster managers, providing the required resources for our application so that we can complete the work that we have defined.

Spark is flexible, and it offers a series of APIs which allow users with different backgrounds to use it. It includes Python, Java, Scala, SQL, and R APIs, with built-in functions and in general a realistically good performance using all of them. It also includes different libraries to handle structured data (Spark SQL), streaming capabilities data (Spark Streaming), machine learning (MLlib), and computation on graphs (GraphX) [176]. Figure 5.28 displays the architecture of the Spark.

¹⁵ "Matplotlib". [Online]. Available: <https://matplotlib.org/> [Last Access: April 2022].

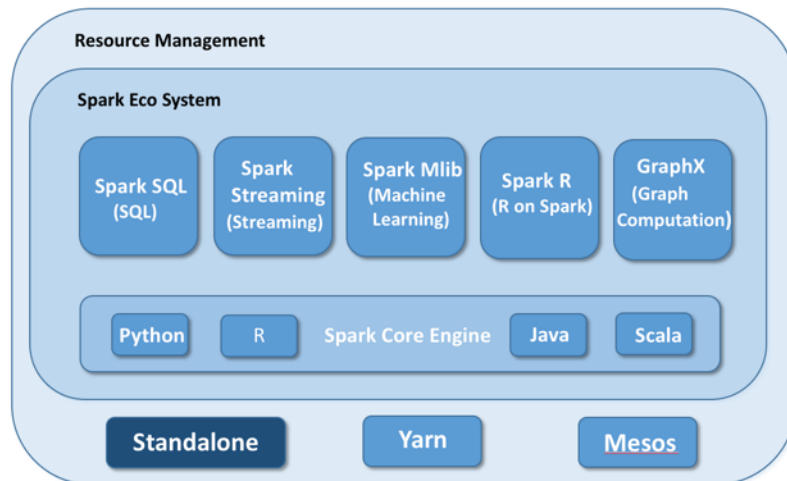


Figure 5. 28: The Ecosystem of the Apache Spark [177].

Apache Spark can be configured and run in four different cluster modes:

- 1- Standalone Cluster
- 2- Apache Mesos
- 3- Hadoop YARN
- 4- Kubernetes

For this research, the standalone cluster has been chosen to implement and configure, since it is a simple cluster manager and an easy to setup cluster. The Architecture of the standalone cluster is based on a master-slave design. To run the Spark application, this architecture contains master and slave nodes. The master node performs as a resource supervisor for the cluster and receives the applications and arranges resources to execute those applications. On the other hand, the worker(s) node is responsible for initiation executors for job execution.

Apache Spark makes it possible to create large-scale machine learning methods that need model parallelism or fundamental data parallelism [178]. Spark Core, which is developed specifically for efficient iterative calculations, is capable of handling these iterative algorithms in an efficient approach. Common activities including model training, model assessment, feature transformation, feature extraction, and tuning are often needed when employing data pipelines and machine learning algorithms for practical applications. To achieve these requirements, Spark's MLib was developed to function as a distributed library for deploying machine learning with the intention of making the development and implementation of such pipelines and algorithms more straightforward. The two primary packages of Spark's MLib are the spark.mlib and spark.ml.32 (Figure 5.29). Spark.ml is based on DataFrames, while spark.mllib is built on RDDs.

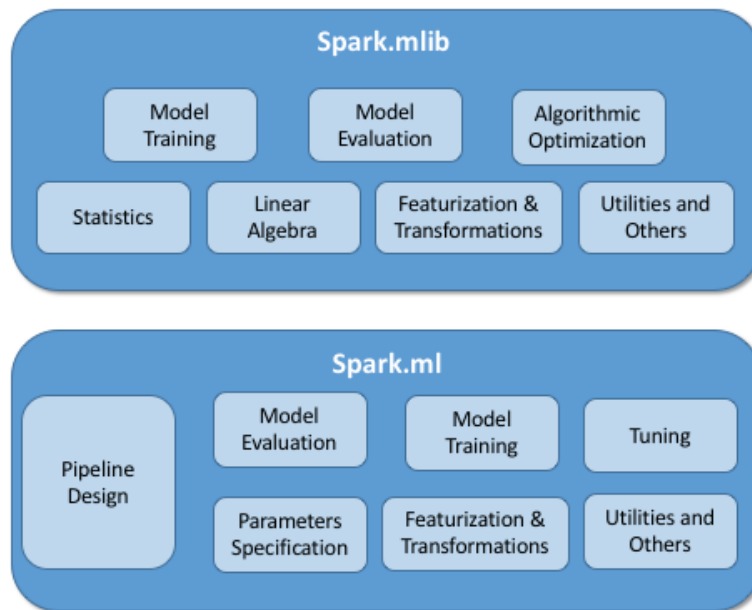


Figure 5. 29: Two main packages of Spark's MLlib library [179].

Each package provides several standard machine learning responsibilities such as model training, transformations, model assessment, featurization, and optimization. While spark.mllib contains packages for statistics, linear algebra, and other fundamental machine learning tools, spark.ml offers the pipelines API for creating, troubleshooting, and fine-tuning machine learning pipelines.

5.7.2.2 Apache Spark Streaming

Apache Spark Streaming [180] is one of the Spark ecosystem components used for this project. It is an extension of the Spark core designed for processing data streams in real time. This data stream can come from different sources like Apache Kafka, Apache Flume, RabbitMQ, Twitter, or Facebook and receive data collected from sensors or devices connected via TCP sockets. Spark Streaming extends the batch processing concept of Apache Spark to stream processing, which decomposes the stream into a series of continuous micro-batches, which can then be manipulated using the Apache Spark API. Thus, batch and stream operations may operate on the same framework and use the same code (in most cases), thereby reducing the overhead for developers and operators. By using Spark streaming for streaming data processing, data can be obtained from many sources, such as Kafka, Flume, Kinesis, or TCP sockets, and then complex algorithms developed by advanced functions (such as Hadoop MapReduce) can be used.

The processed data can be sent to the file system, the database, and the dashboard in real-time. Also, the Spark-provided machine learning (Using Spark MLlib Library) and graphics processing algorithms (Using Spark Graph Library) can be applied to the data streams (Figure 5.30) which is an extension of the source [181]). The criticism of the Spark Streaming approach is that in scenarios that require low-latency responses to incoming data, micro-batch processing may not be comparable to the performance of other streaming media-supporting frameworks (such as Apache Storm, Apache Flink, and Apache Apex). These frameworks all use pure streaming methods instead of micro-batch processing.

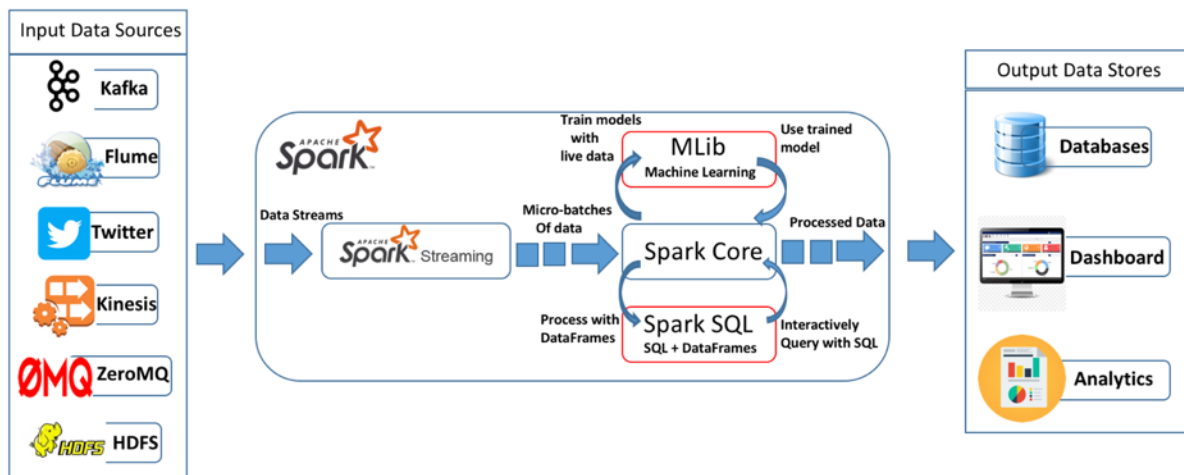


Figure 5. 30: Integration of Apache Streaming with any other Spark components [181].

In this research, Spark Streaming has been used because it is capable of using different formats such as CSV, TEXT, and JSON for reading and writing from/to Kafka topic.

5.7.2.3 Apache Kafka

Apache Kafka [182] is a fault-tolerant, horizontally scalable, distributed data transmission system. It allows users to transfer data in real-time using the publish/subscribe messaging technique. Kafka was created by LinkedIn and currently is an open-source project supported by Confluent, a company that is managed by Apache. Its principal features are as follows: Kafka is an intermediary-based solution that functions by controlling data streams as records in a server cluster. It is possible to store the records (message streams) of numerous server instances in a topic across several data centers using Kafka servers. The records and messages that are kept in a topic are organized into a sequence of tuples. Tuples are immutable Python objects that include a key, a value, and a timestamp.

Kafka provides a Publish-subscribe-based messaging system for the data streams, performing similarly to other message queuing technologies but with excessive performance and gaining very low latency in the transmission of the messages. For scaling up the processing, it offers to users the capability of distributing the data processing into various consumer instances. In addition, it allows for storing streams and replicating them to provide fault tolerance. Kafka permits producers to anticipate recognition so that a deed is not complete unless it is completely reproduced and its persistence is assured. Figure 5.31 displays briefly the architecture of Apache Kafka.

One of the open-source messaging systems with the fastest market growth is Apache Kafka. This is mostly because the architecture design pattern offers distributed systems an effective logging method. Apache Kafka is designed for real-time log streaming, which is ideal for applications with the following requirements [183]:

- Reliable communication between various components.
- Flexible messaging workloads, which may be adapted to changing application needs.

- Streaming data processing in real-time.
- Data/message replay is natively supported.

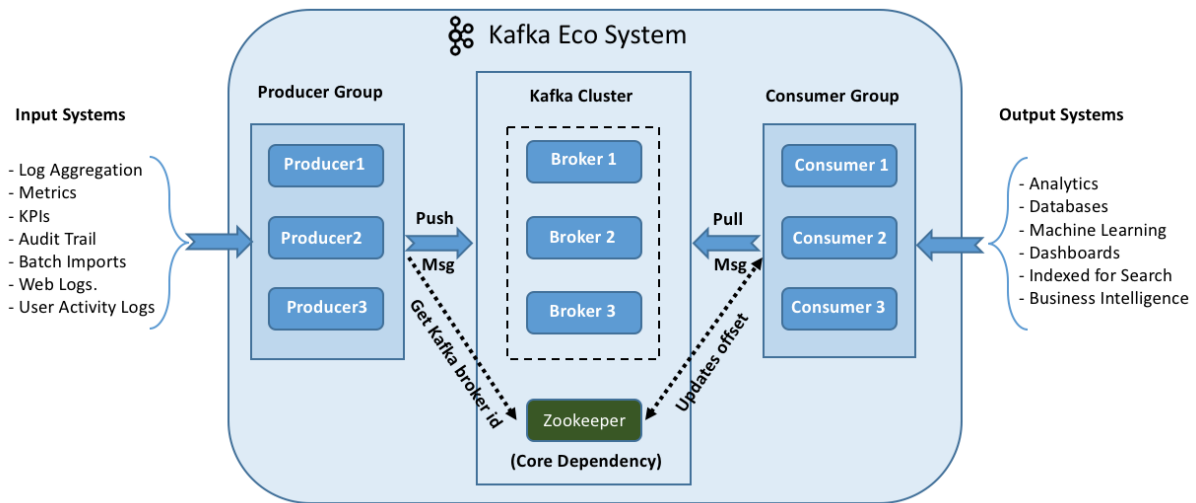


Figure 5. 31: Kafka Architecture [Is an extension of resources [It is an extension of the sources [184] and [185]]

5.7.2.4 Zookeeper

One of the intermediate affiliations of Apache Kafka is Apache Zookeeper [186], which is a service for distributed synchronization. Zookeeper is a service that performs as a coordinating interface between Kafka brokers and consumers. Kafka stores Metadata such as information about topics, brokers and consumers and offsets, etc. in zookeeper. In addition, A huge number of hosts may be efficiently managed with the help of Zookeeper, a distributed coordination service. With its straightforward design, Zookeeper has been able to simplify the complicated task of administering and coordinating a service in a distributed context. It should be noted that Zookeeper gives programmers the ability to concentrate on the logical aspects of their program without being concerned about the distributed nature of the program. Zookeeper is used to manage and coordinate Kafka brokers. The Zookeeper service is used to provide information to both customers and producers as well as the existence of any new broker in the Kafka system or the failure of the Kafka system. In this way, producers and consumers decide to use another broker to coordinate their work.

5.7.2.5 Elasticsearch

Elasticsearch [187] It is a full-text search and analysis engine that is extremely scalable and distributed, and it gives you the ability to store, search, and analyze enormous volumes of data in close to real-time. Through a robust aggregation mechanism and data storage, it may be utilized as an analytical framework even if it is mostly employed as a search engine. Elasticsearch has a comprehensive JSON-based query DSL, demonstrating how easy it is for Lucene to read and write queries. The JSON data storage format is used by most of the NoSQL data stores due to its ability to be very condensed, versatile, and simple to comprehend. From another point of view, this is a document database setting in which semi-structured and structured data can be efficiently retrieved, stored, and documented. All data in the software

settings are stored in JSON file format. Using the JSON format, Elastic Search has its domain-oriented query language. In addition, this setting allows for nesting level queries as needed. REST API is used to expose the function of elastic search settings, Elasticsearch provides a stable environment for storing large amounts of data and content. Most importantly, the technology allows extremely fast data retrieval and storage procedures.

5.8 Implementing the Big Data Pipeline

As mentioned earlier, in the BFDD-S framework the process of the attack detection will be done out of the controller. To provide a scalable, fast, and real-time detection phase, the big data pipeline has been implemented. The big data pipeline consists of Apache Kafka for message queuing, and Apache Spark for real-time data streaming and machine learning processing, in this section, first these tools will be introduced briefly and then explain how the pipeline has been implemented.

5.8.1 Pre-Requisite Configuration

In this research, the Linux Ubuntu Server is used as an operating system of the Server. Therefore, before installing all these tools, it is necessary to make sure that Java 8 is installed on the server, to install Java 8 the following commands should be followed:

```
sudo apt-get update
```

```
sudo apt install openjdk-8-jdk openjdk-8-jre
```

After the installation of Java is completed, the JAVA_HOME and JRE_HOME environment variables should be set. Multiple java applications use these variables to find the JAVA libraries during the runtime

```
cat >> /etc/environment <<EOL
```

```
JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

```
JRE_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre
```

```
EOL
```

5.8.2 Implementation of Big Data Infrastructure

In this section, we describe deeply the implementation process of our intended big data infrastructure. As stated in the previous section, we utilize three big data tools for data pipeline infrastructure, Apache Kafka for message queuing, Apache Spark for data processing, and Elasticsearch for data storing. Therefore, in the following, we explain the deployment of each one of these tools in our infrastructure.

5.8.2.1 Apache Spark Implementation in Standalone Mode

For starting the Spark process and launching a standalone cluster manually, after downloading and installing the Apache Spark, first, a standalone server (Master) should be run by executing the following command:

```
$SPARK_HOME/sbin/start-master.sh
```

After performing the above-mentioned command, figure 5.32 presents the result.

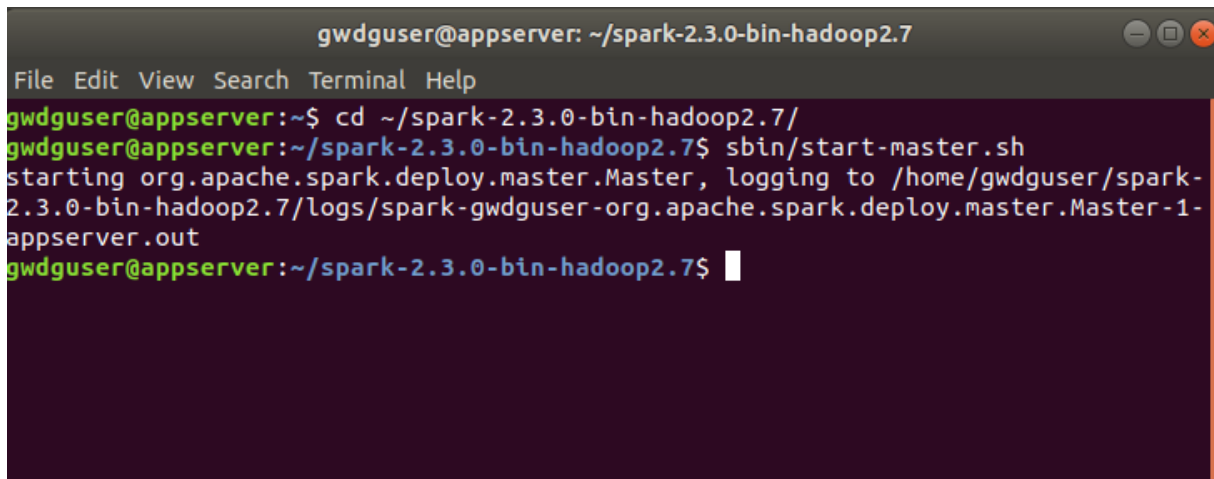


Figure 5. 32: Running Spark in Standalone Server Mode.

After successfully running the command, the result can be seen in the Spark web UI at the address: <http://localhost:8080> (Figure 5.33)

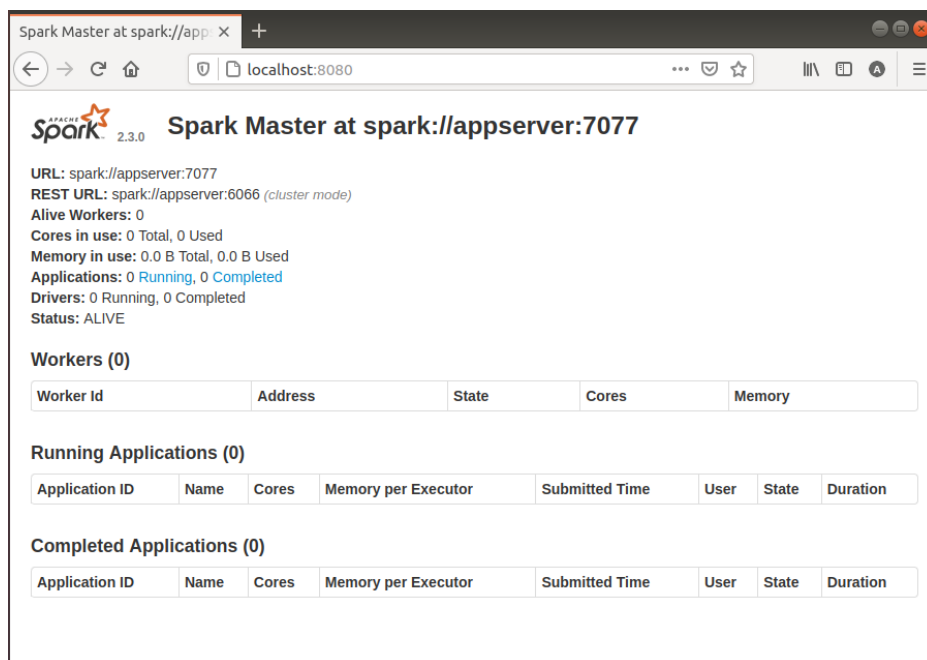


Figure 5. 33: Spark Web GUI.

After completing the previous step, the worker process should be launched. The executors are launched by the worker process for job execution. These executors are where the real data processing take place. One or more workers can be started and connected to the master using the following command:

```
$SPARK_HOME/sbin/start-slave.sh spark://[master Spark server name]:7077
```

Figure 5.34 shows the result after executing the above command.

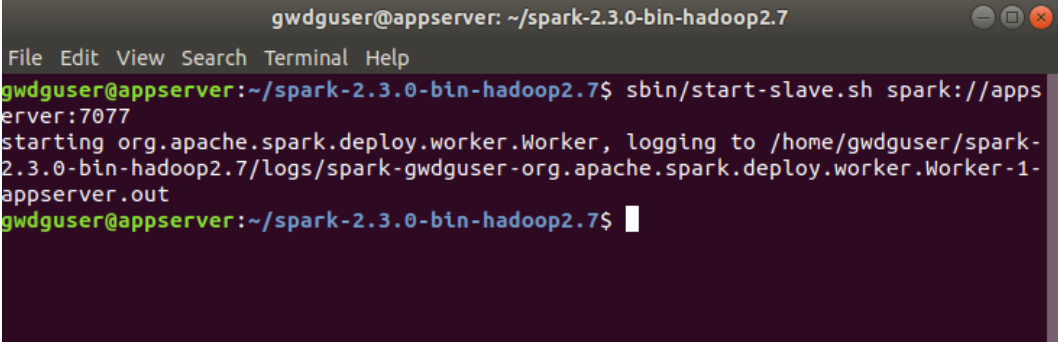


Figure 5. 34: Launching Spark Worker

As figure 5.35 presents, the result can be seen in the master web console:

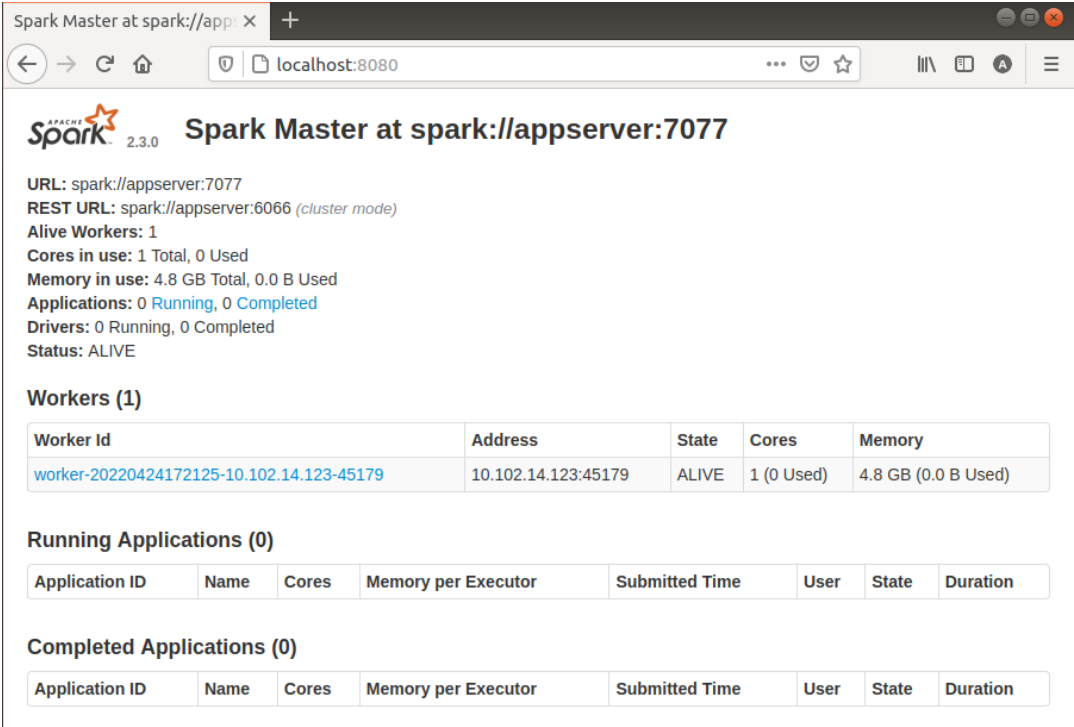


Figure 5. 35: Spark Standalone Mode Information

Apache Spark offers a functionality to interactively develop Scala programs which is called Spark Shell. This functionality is highly useful for both learning the Spark API and interactively analyzing data.

Run the following command to start spark-shell in local mode:

```
$SPARK_HOME/bin/spark-shell spark://[Spark server name]:7077
```

The figure 5.36 shows Spark shell environment.



```
gwdguser@appserver:~/spark-2.3.0-bin-hadoop2.7$ bin/spark-shell spark://appserver:7077
2022-04-24 17:26:17 WARN Uutils:66 - Your hostname, appserver resolves to a loopback address: 1
27.0.1.1; using 10.102.14.123 instead (on interface enp0s3)
2022-04-24 17:26:17 WARN Uutils:66 - Set SPARK_LOCAL_IP if you need to bind to another address
2022-04-24 17:26:18 WARN NativeCodeLoader:62 - Unable to load native-hadoop library for your p
latform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://10.102.14.123:4040
Spark context available as 'sc' (master = local[2], app id = local-1650817590836).
Spark session available as 'spark'.
Welcome to

 version 2.3.0

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_282)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Figure 5. 36: Spark Shell

5.8.2.2 Apache Kafka Implementation

As mentioned earlier, Apache Kafka is a publish-subscribe messaging system. Therefore, for publishing messages it requires the two following entities:

- The producer for facilitating the publication of data and records on topics, and
- The consumer for reading data and messages from the topic.

The steps of the pub-sub messaging workflow are as follows:

- 1- Producers send messages to topics at regular intervals.
- 2- Kafka brokers store all the messages into the configured unit for a specific topic and share the messages evenly between the partitions.
- 3- Consumer shares personal topics. When a topic is shared by a Consumer, Kafka prepares the current topic offset for the Consumer, and the offset is also stored in the zookeeper.
- 4- Consumer requests new messages from Kafka at regular intervals.
- 5- When Kafka receives messages from the producer, it sends these messages to the consumer.
- 6- The consumer receives and processes messages.
- 7- When the message is processed, the consumer sends a confirmation to the broker.
- 8- Kafka modifies the offset value and updates zookeeper after receiving confirmation. Hence the offsets are stored in the zookeeper, the consumer can read the next messages correctly and the above steps are repeated until the consumer stops the request.

Consumer has options that can go back in time and the desired topic goes back and messages can be read.

After successfully downloading and installing Apache Kafka, to make Kafka run, it is required to update and set a few properties in the Kafka configuration file which is available at the following address:

```
$KAFKA_HOME/config/server.properties
```

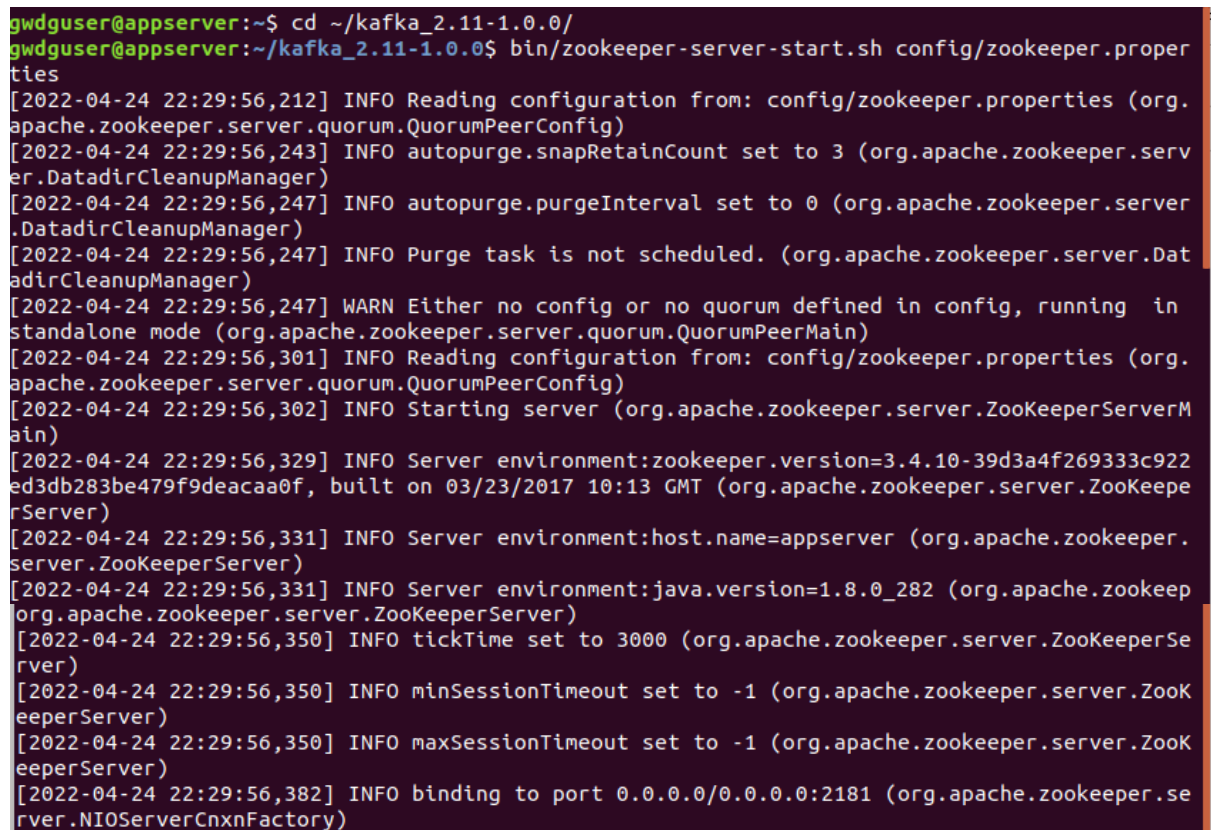
The following two lines must be added to the Kafka configuration file:

```
listeners=PLAINTEXT://[Name of the Server]:9092
advertised.listeners=PLAINTEXT://[Name of the Server]:9092
delete.topic.enable=true
```

Listeners are the addresses to which the socket server listens, and advertised.listener is the hostname and port number of the broker which will broadcast to producers and consumers. After editing the Kafka configuration file, and before launching the Kafka application, first, the Zookeeper tool must be run.

In this step, the following command is used to launch the Zookeeper application (Figure 5.37):

```
$KAFKA_HOME/bin/zookeeper-server-start.sh config/zookeeper.properties
```



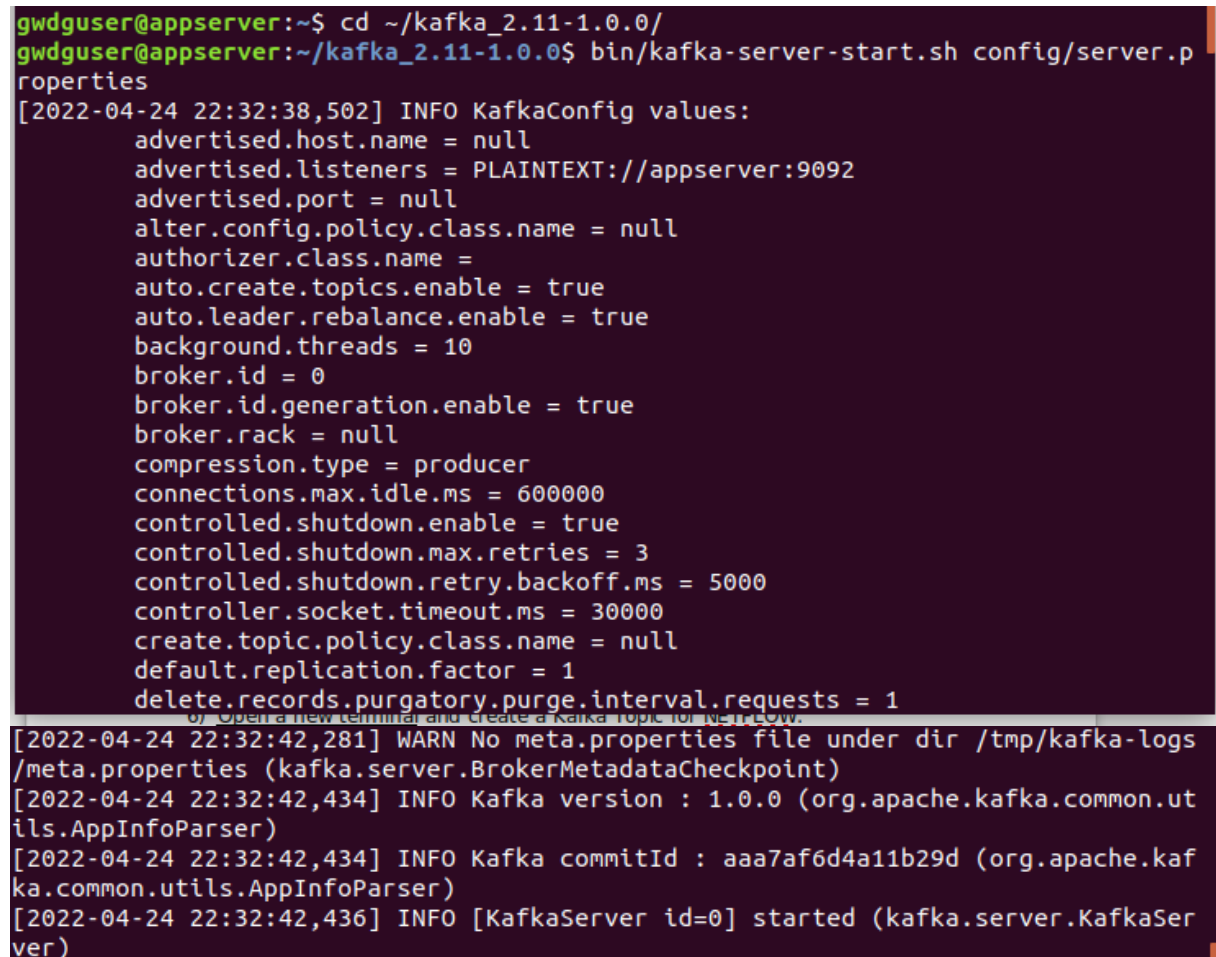
```
gwdguser@appserver:~$ cd ~/kafka_2.11-1.0.0/
gwdguser@appserver:~/kafka_2.11-1.0.0$ bin/zookeeper-server-start.sh config/zookeeper.properties
[2022-04-24 22:29:56,212] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-04-24 22:29:56,243] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DataDirCleanupManager)
[2022-04-24 22:29:56,247] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DataDirCleanupManager)
[2022-04-24 22:29:56,247] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DataDirCleanupManager)
[2022-04-24 22:29:56,247] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMain)
[2022-04-24 22:29:56,301] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-04-24 22:29:56,302] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2022-04-24 22:29:56,329] INFO Server environment:zookeeper.version=3.4.10-39d3a4f269333c922ed3db283be479f9deacaa0f, built on 03/23/2017 10:13 GMT (org.apache.zookeeper.server.ZooKeeperServer)
[2022-04-24 22:29:56,331] INFO Server environment:host.name=appserver (org.apache.zookeeper.server.ZooKeeperServer)
[2022-04-24 22:29:56,331] INFO Server environment:java.version=1.8.0_282 (org.apache.zookeeper.org.apache.zookeeper.server.ZooKeeperServer)
[2022-04-24 22:29:56,350] INFO tickTime set to 3000 (org.apache.zookeeper.server.ZooKeeperServer)
[2022-04-24 22:29:56,350] INFO minSessionTimeout set to -1 (org.apache.zookeeper.server.ZooKeeperServer)
[2022-04-24 22:29:56,350] INFO maxSessionTimeout set to -1 (org.apache.zookeeper.server.ZooKeeperServer)
[2022-04-24 22:29:56,382] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)
```

Figure 5. 37: Running Zookeeper

After running the zookeeper successfully, the Kafka server (As the Kafka Broker) can be launched with the following command:

```
$KAFKA_HOME/bin/kafka-server-start.sh config/server.properties
```

Figure 5.38 presents the result of executing the above command.



```
gwdguser@appserver:~$ cd ~/kafka_2.11-1.0.0/
gwdguser@appserver:~/kafka_2.11-1.0.0$ bin/kafka-server-start.sh config/server.p
roperties
[2022-04-24 22:32:38,502] INFO KafkaConfig values:
  advertised.host.name = null
  advertised.listeners = PLAINTEXT://appserver:9092
  advertised.port = null
  alter.config.policy.class.name = null
  authorizer.class.name =
  auto.create.topics.enable = true
  auto.leader.rebalance.enable = true
  background.threads = 10
  broker.id = 0
  broker.id.generation.enable = true
  broker.rack = null
  compression.type = producer
  connections.max.idle.ms = 600000
  controlled.shutdown.enable = true
  controlled.shutdown.max.retries = 3
  controlled.shutdown.retry.backoff.ms = 5000
  controller.socket.timeout.ms = 30000
  create.topic.policy.class.name = null
  default.replication.factor = 1
  delete.records.purgatory.purge.interval.requests = 1
[2022-04-24 22:32:42,281] WARN No meta.properties file under dir /tmp/kafka-logs
/meta.properties (kafka.server.BrokerMetadataCheckpoint)
[2022-04-24 22:32:42,434] INFO Kafka version : 1.0.0 (org.apache.kafka.common.ut
ils.AppInfoParser)
[2022-04-24 22:32:42,434] INFO Kafka commitId : aaa7af6d4a11b29d (org.apache.kaf
ka.common.utils.AppInfoParser)
[2022-04-24 22:32:42,436] INFO [KafkaServer id=0] started (kafka.server.KafkaSer
ver)
```

Figure 5. 38: Running Kafka Server (Broker)

After completing the installation and running the Zookeeper and Kafka, first, a new Topic should be defined to store the events as shown in the following command:

```
$KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper localhost: 2181 --replication-factor 1 --partitions 1 --topic EXPTINFO
```

In this command, the replication-factor option specifies the number of copies of data that will be generated. Because only one single instance is running in this research, this value is set to 1.

Additionally, in the above command, the partition option indicates the number of brokers to which the data are split. In this experiment, the single broker is used therefore the number in the partition option is set to 1(Figure 5.39).

```

gwdguser@appserver:~$ cd ~/kafka_2.11-1.0.0/
gwdguser@appserver:~/kafka_2.11-1.0.0$ bin/kafka-topics.sh --create --zookeeper
localhost:2181 --replication-factor 1 --partitions 1 --topic EXPTINFO
Created topic "EXPTINFO".
gwdguser@appserver:~/kafka_2.11-1.0.0$ █

```

Figure 5. 39: Create a Topic in Kafka

After executing the above command, the created Topic can be checked by running the following command (See figure 5.40):

```
$KAFKA_HOME/bin/kafka-topics.sh --list --zookeeper localhost:2181
```

```

gwdguser@appserver:~/kafka_2.11-1.0.0$ bin/kafka-topics.sh --list --zookeeper lo
calhost:2181
EXPTINFO
gwdguser@appserver:~/kafka_2.11-1.0.0$ █

```

Figure 5. 40: Kafka Topic Information

In the old version of Kafka, it was necessary for the Kafka consumer to communicate directly with Zookeeper, but in the new version of Kafka, the Zookeeper is deprecated and replaced with the bootstrap-server to provide the connectivity between the Kafka consumer and Kafka broker. As can be seen in figure 5.41.

Kafka uses the command line to define consumers for reading data from the Kafka cluster as shown in the below command:

```
$KAFKA_HOME/bin/kafka-console-consumer.sh --zookeeper localhost: 2181 --topic EXPTINFO
```

```

gwdguser@appserver:~$ cd ~/kafka_2.11-1.0.0/
gwdguser@appserver:~/kafka_2.11-1.0.0$ bin/kafka-console-consumer.sh --zookeeper
localhost:2181 --topic EXPTINFO
Using the ConsoleConsumer with old consumer is deprecated and will be removed in
a future major release. Consider using the new consumer by passing [bootstrap-s
erver] instead of [zookeeper].
█

```

Figure 5. 41: Define Kafka Consumer

5.8.2.3 Elasticsearch Implementation

After completing the installation and configuring the Apache Kafka and Apache Spark, the Elasticsearch for storing data should be installed and run.

After Installing, for running Elasticsearch simply the following command should be executed:

```
$ELASTICSEARCH_HOME/bin/elasticsearch
```

The result of running above command is shown in figure 5.42.

```

gwdguser@appserver:~$ cd ~/elasticsearch-6.2.3/
gwdguser@appserver:~/elasticsearch-6.2.3$ bin/elasticsearch
[2022-04-26T12:39:17,346][INFO ][o.e.n.Node                ] [] initializing ...
[2022-04-26T12:39:18,074][INFO ][o.e.e.NodeEnvironment ] [Lpi0Std] using [1] data paths, mounts
  [[/ (/dev/sda1)], net usable_space [31.6gb], net total_space [48.9gb], types [ext4]
[2022-04-26T12:39:18,075][INFO ][o.e.e.NodeEnvironment ] [Lpi0Std] heap size [1015.6mb], compre
  ssed ordinary object pointers [true]
[2022-04-26T12:39:18,392][INFO ][o.e.n.Node                ] node name [Lpi0Std] derived from node
  ID [Lpi0StdQqBe9zvOfzsomrQ]; set [node.name] to override
[2022-04-26T12:39:18,393][INFO ][o.e.n.Node                ] version[6.2.3], pid[13961], build[c59f
  f00/2018-03-13T10:06:29.741383Z], OS[Linux/5.4.0-72-generic/amd64], JVM[Private Build/OpenJDK 64-B
  it Server VM/1.8.0_282/25.282-b08]
[2022-04-26T12:39:18,394][INFO ][o.e.n.Node                ] JVM arguments [-Xms1g, -Xmx1g, -XX:+Us
  eConcMarkSweepGC, -XX:CMSInitiatingOccupancyFraction=75, -XX:+UseCMSInitiatingOccupancyOnly, -XX:+
  AlwaysPreTouch, -Xss1m, -Djava.awt.headless=true, -Dfile.encoding=UTF-8, -Djna.nosys=true, -XX:-Om
  itStackTraceInFastThrow, -Dio.netty.noUnsafe=true, -Dio.netty.noKeySetOptimization=true, -Dio.nett
  y.recycler.maxCapacityPerThread=0, -Dlog4j.shutdownHookEnabled=false, -Dlog4j2.disable.jmx=true, -
  Djava.io.tmpdir=/tmp/elasticsearch.Bh2mQfU4, -XX:+HeapDumpOnOutOfMemoryError, -XX:+PrintGCDetails,
  -XX:+PrintGCDateStamps, -XX:+PrintTenuringDistribution, -XX:+PrintGCApplicationStoppedTime, -Xlog
  gc:logs/gc.log, -XX:+UseGCLogFileRotation, -XX:NumberOfGCLogFiles=32, -XX:GCLogFileSize=64m, -Des.
  path.home=/home/gwdguser/elasticsearch-6.2.3, -Des.path.conf=/home/gwdguser/elasticsearch-6.2.3/co
  nfig]
[2022-04-26T12:39:30,422][INFO ][o.e.n.Node                ] [Lpi0Std] started
[2022-04-26T12:39:33,930][INFO ][o.e.g.GatewayService       ] [Lpi0Std] recovered [8] indices into c
  luster_state
[2022-04-26T12:40:12,110][INFO ][o.e.c.r.a.AllocationService] [Lpi0Std] Cluster health status chan
  ged from [RED] to [YELLOW] (reason: [shards started [[my_index][0]] ...]).

```

Figure 5. 42: Running Elasticsearch

The Elasticsearch can be configured through its main configuration file called elasticsearch.yml which is stored in the /etc/elasticsearch directory. It should be considered that the Elasticsearch configuration file is in YAML format which means that the indentation should be maintained. By default, Elasticsearch listens for the network traffic on port 9200. The Elasticsearch uses RESTful API for communication which means that it responds to the typical CRUD commands (CRUD stands for **C**reate, **R**ead, **U**ppdate, and **D**elate). Therefore, for checking the successful running Elasticsearch on port 9200, the curl command and GET request can be used:

```
$curl -X GET 'http://localhost:9200'
```

The following figure 5.43 displays the result.

```

gwdguser@appserver:~$ curl -X GET 'http://localhost:9200'
{
  "name" : "Lpi0Std",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "75XaPvuxQD2q8s1f8wZQjg",
  "version" : {
    "number" : "6.2.3",
    "build_hash" : "c59ff00",
    "build_date" : "2018-03-13T10:06:29.741383Z",
    "build_snapshot" : false,
    "lucene_version" : "7.2.1",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
gwdguser@appserver:~$ █

```

Figure 5. 43: Elasticsearch Information

It should be noticed that Elasticsearch is a distributed and very simple-to-scale NoSQL database, since it is built on top of the Lucene engine and uses the HTTP interface. Being a NoSQL database means that it is not necessary to have any structured data, and it does not employ any standard structured query language while searching for information.

For configuring Elasticsearch, first, an index should be defined, the index is a group of documents that have comparable features. In other words, indexes are groups of associated JSON files that can be considered as the base unit of storage in Elasticsearch. Figure 5.43 indicates the index file which is created in this research for storing the network traffic information received, aggregated, transformed, and sent from the SDN controller to the data pipeline for processing. As mentioned earlier, in the “Machine Learning” section of this chapter, and “Data set Selection” sub-section, in this framework prototype, for DDoS attack detection, we consider four traffic features, Source Address, Destination Address, Destination Port, and Sum of the Flows.

```
def create_index(es_obj, index_name):
    created = False
    settings = {
        "settings": {
            "number_of_shards": 1,
            "number_of_replicas": 0
        },
        "mappings": {
            "_doc": {
                "properties": {
                    "date": {
                        "type": "date",
                        "format": "yyyy-MM-dd"
                    },
                    "SrcAddr": {"type": "integer"},
                    "DstAddr": {"type": "integer"},
                    "DstPort": {"type": "integer"},
                    "SumOfFlows": {"type": "long"}
                }
            }
        }
    }
    try:
        if not es_obj.indices.exists(index_name):
            es_obj.indices.create(index=index_name, ignore=400, body=settings)
            print('Created Index')
            created = True
    except Exception as ex:
        print(str(ex))
    finally:
        return created
```

Figure 5. 44: Create Index in Elasticsearch for Storing Data

Therefore, for this research, the index called “trafficinfo” is created to store the network traffic received from Apache Kafka for later processing by Apache Spark-based machine learning. To ensure that the index is properly created in Elasticsearch, the following command can be used (Figure 5.45 shows the result).

```

gwdguser@appserver:~$ curl 'localhost:9200/trafficinfo/_mapping?pretty=true'
{
  "trafficinfo" : {
    "mappings" : {
      "entry" : {
        "properties" : {
          "DstAddr" : {
            "type" : "integer"
          },
          "DstPort" : {
            "type" : "integer"
          },
          "SrcAddr" : {
            "type" : "integer"
          },
          "SumofFlows" : {
            "type" : "long"
          }
        }
      }
    }
  }
}
gwdguser@appserver:~$

```

Figure 5. 45: Information of the Created Index in Elasticsearch

In the following section, how to implement the data pipeline is explained, when all necessary tools and applications are properly installed and configured,

5.8.3 Implementing Data Pipeline

For implementing the data pipeline two Python files are created, one for providing the connection between Apache Kafka, and Apache Spark and the second one for implementing a machine learning model into Apache Spark using the Mllib library. In the initial phase, the essential modules for different purposes such as configuring Spark, defining Streaming, and enabling Kafka to Spark Streaming must be imported. In addition, it is required to import the JSON module for processing the incoming data and also the Elasticsearch module (Figure 5.46).

```

#      Importing Spark module
from pyspark import SparkConf, SparkContext
#      Importing Spark Streaming Module
from pyspark.streaming import StreamingContext
#      Importing Kafka Module
from pyspark.streaming.kafka import KafkaUtils
#      Importing ElasticSearch Module
from elasticsearch import Elasticsearch
#      Importing JSON for Parsing the Data
import json

import datetime

```

Figure 5. 46: Import Spark Essential Libraries

Then the Spark Context must be created. Figure 5.47 illustrates the necessary steps to accomplish this object. The Spark Context is the primary point for accessing Spark capabilities. Creating RDDs, accumulators, and broadcast variables on a Spark cluster is accomplished via the usage of the Spark Context object, which establishes a link to a Spark cluster [188].

```
conf = SparkConf()
conf.setAppName("Python SparkStreaming Kafka")
conf.setMaster('local[*]')
sc = SparkContext(conf=conf)
sc.setLogLevel("WARN")
n_secs = 10
ssc = StreamingContext(sc, n_secs)
```

Figure 5. 47: Create Spark Context

The setMaster ('local [*]') allows Spark to be run locally with as many worker threads as logical cores available on the computer.

It is not essential to use the setLogLevel function, but it is beneficial for reducing the amount of noise on stdout (Standard Output) that would otherwise obscure the real outputs from the task.

Then an instance of the Streaming Context object (available as variable sc) must be constructed, which acts as a doorway for the streaming to begin. The Spark Streaming divides the incoming data into batches and creates mini-batch RDDs (RDD is Spark's fundamental abstraction), and the output is likewise produced in batches. Therefore, the batch interval must be specified when a Streaming Context object is created. This guarantees that the streaming data is separated into batches depending on the time slice that is being used.

```
n_secs = 10
```

```
scc = StreamingContext (sc, n_secs)
```

By using a batch interval, Spark is informed how long it should wait before fetching the data. To achieve optimal processing performance, it is usually suggested to choose a low batch size interval value. In this framework, the batch interval is set to 10 seconds. The figure 5.48 clarifies the process of dividing the live input data into batches and then processing by the Spark engine.



Figure 5. 48: Process of the live input data in the Spark streaming [189].

To provide a connection to the Kafka cluster, the streaming context is used. In this step, the input stream is created to pull messages from the Kafka Brokers. Some parameters should be specified for creating the input stream:

- Zookeeper's hostname and port number that connects from this stream.
- Name of the Topic (here is EXPTINFO) and the number of partitions this stream consumes in parallel (Figure 5.49).

```
kafkaStream = KafkaUtils.createStream(ssc, "localhost:2181",
                                     "spark-streaming-consumer",
                                     {"EXPTINFO":1})
lines = kafkaStream.map(lambda x: json.loads(x[1]))
```

Figure 5. 49: Generates an input stream that fetches messages from the Kafka broker.

Using the above-mentioned steps, the connection between Kafka broker and Spark streaming will be provided. Then, to complete the data pipeline two functions are written, one for sending and storing the network traffic from Kafka to Elasticsearch (called StoringtoES) and the other one for retrieving the stored information from Elasticsearch (called FetchfromES) into Apache Spark for the data processing phase.

5.8.4 Implementing Machine Learning in Apache Spark

Regularly, for implementing and creating a machine learning model in Python, as an input for training, raw DataFrame is used, particularly when working with the Scikit-learn library but in a distributed environment the situation might be more challenging because in this condition for preparing the training data the Assemblers should be used. Therefore, in this step, a simple Spark DataFrame should be established. DataFrame is a Distributed set of data organized in registered columns. In fact, its concept can be considered as corresponding to a table in a relational database. DataFrames can be created from an extensive range of sources such as external databases, structured data files, hive tables, or existing RDDs.

Spark is based on the concept of a fault-tolerant collection of components that may be processed in parallel and is known as a resilient distributed dataset (RDD) [190].

For creating RDDs in Spark, there are two methods, one method is to use an external data source such as HDFS, File system, etc., and load it into the system and the other method which is used in this research is to consider an existing group of data and perform the transformation operation by utilizing the parallelize () method of SparkContext.

The first item that must be constructed in order to utilize the parallelize () function is a SparkContext object; as cited earlier in this section then, the following commands, as shown in figure 5.50, can be used to create a data frame.

```
# Create RDD from stored Information in ElasticSearch
myRDD = sc.parallelize(tempdata.tolist())

# Convert created RDD to the Data Frame.
df = spark.createDataFrame(myRDD)
```

Figure 5. 50: Creating RDD in Spark

In the previous section, the process of evaluating different machine learning methods for detecting DDoS attacks in the network using two popular data sets was described in detail and the Random Forest algorithm has been chosen for implementation in the data processing phase of the data pipeline of the BFDD-S framework using Apache Spark. In order to implement the machine learning algorithm into the framework using Apache Spark MLlib library, the following methods are utilized for the pre-processing phase in Spark approaches:

1) VectorAssembler

2) Scaling and normalizing

Vectors are utilized in algorithms and processes in the field of machine learning to describe the objective variable while training an algorithm. Therefore, by using the Spark ML library, we can take advantage of using the VectorAssembler module, which provides the possibility for transforming numerical features into a single vector that can be later passed to the machine learning models (Figure 5.51 shows the relevant commands).

```
# Converting the features to a vector for implementing
# Random Forest Algorithm in the Spark.
Assembler = VectorAssembler(
    inputCols=FEATURE_COLS, outputCol="features")
```

Figure 5. 51: Transforming Numerical Features into Single Vector

As another step of pre-processing of the implementation of the machine learning model to the BFDD framework, we perform the scaling and normalization task. This task is not mandatory but, extremely suggested that before employing the machine learning model for preventing the possibility of an algorithm being unresponsive to particular features and data leakage, it is an appropriate approach to scale and normalize the features [191].

As Figure 5.52 shows for scaling and normalization of the features task in Spark, there is a class called “StandardScaler” in the ML library [192].

```
# Scaling the features.
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures"
    withStd=True, withMean=False)
```

Figure 5. 52: Scaling Features in Spark

and finally, for the machine learning model, after loading the appropriate module from the ML library with the commands displayed in figure 5.53.

```
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StandardScaler
from pyspark.sql import SparkSession
import numpy as np
```

Figure 5. 53: Import Random Forest Classifier in Spark

the RandomForestClassifier is used to define the machine learning model (See figure 5.54).

```
# RandomForest Classifier Model
rf = RandomForestClassifier(featuresCol='scaledFeatures', labelCol='LabelIndex',
                           numTrees=30)
rfModel = rf.fit(df_scaled)
```

Figure 5. 54: Define Random Forest Model in Spark

As the figure shows, the Random Forest Classifier method is comprised of some parameters. This algorithm requires defining the feature column, created by the VectorAssembler module as mentioned earlier in this section, as well as the dataset's label column, and the number of trees is which in this model defined as 30 without tuning the hyperparameters.

In this research, the major concern is DDoS attack detection, and accuracy should be considered as a vital criterion for any intrusion detection approach. Therefore, for creating a Random Forest model with high accuracy, the number of trees plays a significant role. In Addition, the number of trees in this model affects the memory utilization and processing time. Some research [193] claimed that if the number of trees has been increased without any logic behind it, the anomaly detection approach may suffer serious consequences. In addition, they demonstrated that for achieving the optimal value considering both processing time and accuracy, the suitable value for trees is around 30 trees.

The next step in this approach is to create and conduct the proper action request to the controller based on the machine learning process for all incoming traffic. As stated in the earlier section, the northbound interface provides communication between the controller and the application layer. There are several different methods that can be used as the Northbound interface, in this framework, the REST API is considered for providing the communication between the big data pipeline and the controller.

When Spark Streaming context is run, then load the stored traffic from the Elasticsearch using FetchfromES () function, and perform data processing phase, if Spark-based machine learning detects any malicious traffic, the attacker device is recognized by the traffic sender's IP address (SrcIPAddr), then a REST API request is created and send to the controller with the IP address of the attacker. To achieve this objective, first, we have to create a REST API channel to the controller in order to send a specific request.

API is a specific service through the web that utilizes the standard HTTP protocols to grant access to the particular data for other applications. There are multiple architecture styles that can be used in designing API such as FALCORE, gRPC, GraphQL, and REST. But probably, the most prominent architectural style of APIs for web services is REST which is the acronym for REpresentational State Transfer. It makes client-server communication easy by providing a set of guidelines. In this particular use case, the REST architecture is implemented to the BFDD-S framework to propose Northbound API to the controller.

the general processes for designing and performing a RET API communication are as follows:

- 1- A client or application to send a request and some data to a specific address (URL). The data will send through the request method enclosed in the message body in JSON format. The popular request methods are GET, POST, and PUT.

JSON (JavaScript Object Notation) is a standard format for sending and receiving data using REST APIs architecture.

- 2- The sever located in the particular address (URL) received the data and will perform some operations on them and reply response to the client or application. The response information should also be always in JSON format.
- 3- The client or application will decide about the received response. The figure 7.48 concisely explains the REST API communication architecture.

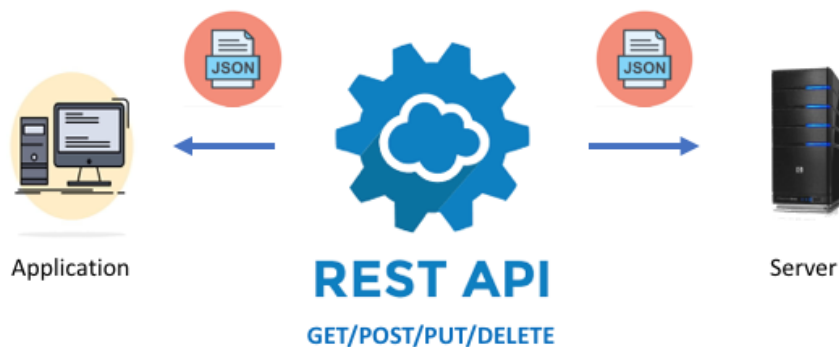


Figure 5. 55: REST API Architecture

For implementing the above process into the framework in order to provide a REST API channel as a northbound interface between the controller and data processing pipeline, first, a block of data (called AttackDevInfo) in JSON format is created with all necessary information such as network device ID (deviceId), type of the traffic (ETH_TYPE), attacker IP address and subnet mask, etc. is created which can be seen in the figure 5.56.

```

AttackDataInfo = {
  "priority": 40000,
  "timeout": 0,
  "isPermanent": "false",
  "deviceId": "of:0000000000000001",
  "treatment": {},
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE", "ethType": "0x0800"
      },
      {
        "type": "IPV4_SRC", "ip": "10.0.0.1/32"
      }
    ]
  }
}

```

Figure 5. 56: JSON File for Creating REST API Request

Then, a REST API POST method is used to create a request object and send a combination of the specific data such as the REST server address (here, the ONOS controller URL), the data (AttackDevInfo), and the authentication information (here, ONOS admin username and password) as shown in the below figure 5.57.

```

Controller_Addr = "http://[CONTROLLER_IP_ADDRESS]:8181/onos/v1/flows/[DEVICE_ID]?appId=org.onosproject.fwd"
response = requests.post(Controller_Addr, data=json.dumps(AttackDataInfo), auth=('onos', 'rocks'))

```

Figure 5. 57: Sending Request to the Controller via REST API POST Method

After receiving and processing the request, the REST API server will provide a response. This response contains an HTTP status code. This code represents the result of the request, and the sender application can check and make a decision based on this received code. The HTTP code has a range between 1xx to 5xx.

If the response code is 200 (which means OK) or 204 (which means No Content), this indicates the successful conclusion of the sent request.

5.9 Chapter Summary

The major goal of this chapter is to find solutions to a few research questions. In this chapter, a robust and resilient intrusion detection framework (BDDF-S) has been presented that can offer a scalable and reliable DDoS attack detection and mitigation system for the SDN environment with a fast, novel, and real-time detection technique. For achieving this purpose, an effective method appropriate to the architecture and characteristics of the SDN network has been proposed. The BDDF-S framework is a heterogeneous system in which the detection phase is done out of the controller and gathering information and mitigation are done by the SDN controller.

In designing the BFDD-S prototype, we utilize a combination of big data tools and machine learning together with taking advantage of the developed SDN controller. The proposed framework consists of three modules, one module for gathering and formatting statistical information, the second module for detecting attacks, and the third module is the machine learning process.

For classifying the network traffic to determine malicious flows more accurately, the third module is defined and implemented to utilize the machine learning algorithm classifier capability offered by Apache Spark MLlib library and implement the Random Forest algorithm into the data processing module based on the results of our evaluation on different machine learning algorithms. The outcome of this part would be an answer to research question number two, which is:

- **Which machine learning classifier algorithm is appropriate for detecting DDoS attacks?**

The attack detection module is located in a server and uses data pipeline infrastructure and machine learning for detecting DDoS attacks. The prevention task will be done by the controller (We used ONOS as a controller for this experience). In this framework for creating data pipeline infrastructure, we use the following big data analytic tools, Kafka for message queuing, Apache Spark for data processing, and Elasticsearch for storing data.

By creating a data processing pipeline with these tools with the following features such as short response time, high efficiency and error tolerance, high scalability, and ability to process fast streams of events of Kafka together with Real-Time Stream Processing, Fault Tolerance, scalability, and the lightning-fast analytic engine of Apache Spark and horizontally scalability, multi-tenancy, and speed of Elasticsearch. The conclusion of this part would be an explanation for research question number three:

- **How to implement a data pipeline to detect malicious network traffic in the network?**

In addition, for information gathering, aggregating the flows, and converting them to an appropriate format we developed an API for the ONOS core controller using Java programming language. The result of this part which is developed an API called Exporter would be an explanation for research question number four which is:

- **How can a detection module be integrated into the SDN controller for the detection and prevention of a DDoS attack in the SDN environment?**

Chapter 6

6 Experimental Setup and Performance Evaluation

In this chapter, we explain the details of the hardware and software that we use for setting up and preparing the experimental test bed. Since our preliminary purpose when designing the BFDD-S framework was to offer a robust, resilient, and fast DDoS attack detection in the SDN network, for evaluating the performance of the framework, we generated normal (ICMP packet) and anomaly traffic (TCP SYN flood) simultaneously and walk through our approach, the network traffic captures via the SDN controller and process through the data pipeline for attack detection and outcomes are stored in the data store. Finally, the diagrams will be used to demonstrate the performance evaluation of the framework. In this research, we consider the CPU consumption, memory utilization, and estimated process time for legitimate traffic as evaluation metrics.

6.1 Setup Experimental Testbed

To evaluate the performance of the BDDF-S framework, we implement a testbed which is depicted in figure 6.1. We design a simple leaf-spine network architecture by using four Dell Power Edge R430 servers and four bare metal EdgeCore AS 4610 switches that support OpenFlow protocol and one HP2530-8G as a management switch. These servers and Switches connect to each other using the Fiber cable. Table 6.1 portrays the hardware and software we use in this research.

For emulating a SDN network architecture, we use this simple but real test bed as an underlay network with the help of virtual Open vSwitches. The emulated SDN network runs within one Linux machine server and is controlled by a remote ONOS SDN controller which is installed and running on another physical server and implements the Big Data pipeline in another physical machine. To emulate the SDN-based network, we use Mininet [169] to define the topology on one server, consisting of one Web server and several hosts which are all connected to an SDN-based switch.

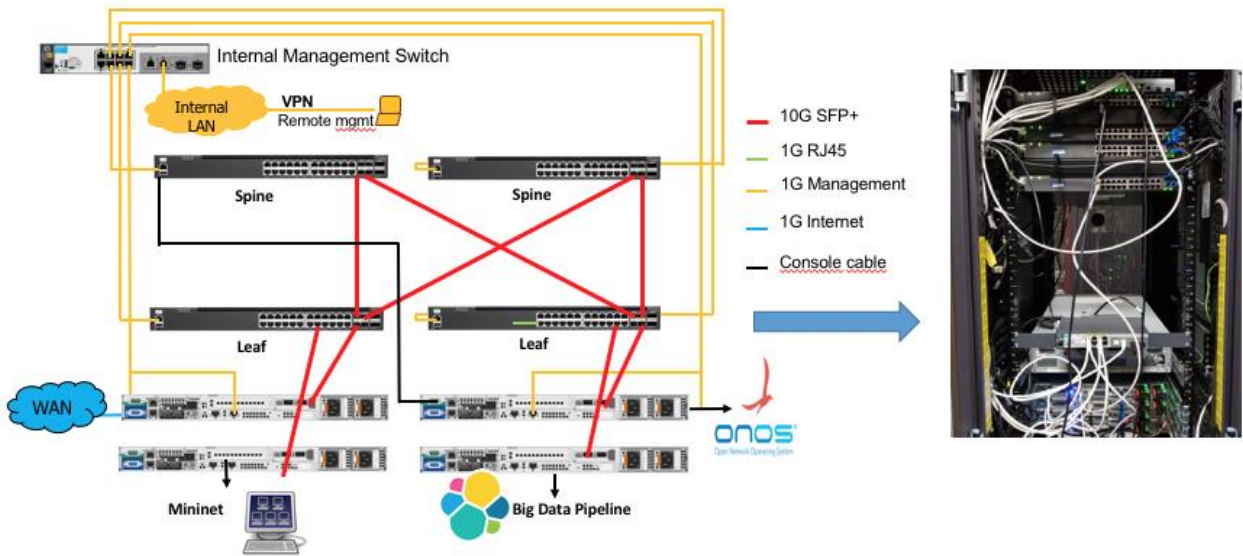


Figure 6. 1: Experimental Testbed Architecture.

Furthermore, one Linux server machine has been used to implement a Bigdata pipeline which is controlled by a remote SDN controller (ONOS) running on another physical server machine, to emulate the SDN based network, we use Mininet to define a network topology on one of server machines, consisting of one web server and several hosts which are all connected to a virtual SDN switch (OpenVswitch).

Table 6. 1: Hardware and Software used in the Testbed.

Experimental Setup	Type	Specification	Details/Version
Hardware	Switch	Edge Core AS4610	24-Ports RJ45, 4x10 GbE SFP+ ONL(Open Network Linux)
		HP2530-8G	12-Ports RJ45
Hardware	Server	DELL Power Edge R430	Intel Xeon Processor 32 GB Memory 500 GB HDD
Platform/Environment	SDN Controller	ONOS	Peacock 1.15.0
	SDN Emulator	Mininet	2.2.2

6.2 Deploying Network Topology

Mininet is capable of creating the network with arbitrary topology in a simple and fast way using the Python API [194]. By using this API, a script consisting of various features of the network can be defined, such as the topology, the type of network devices, the IP addresses of the controller, the communication protocol, etc.

Therefore, for deploying and emulating our network topology for our experimental testbed using Mininet, we use the Mininet Python API to develop a Python script.

In the script, after importing all essential libraries, the network devices such as hosts and switches are specified using the following commands:

For adding a host to the topology, the following command is used:

```
host = self.addHost('h')
```

For adding a switch to the topology, the following command is used:

```
switch = self.addSwitch('s')
```

For defining a link between two devices, the following command is used:

```
self.addLink(device1,device2)
```

Once the topology of the network has been defined in the script, the type of switch used in the network, the IP address of the SDN controller, and the protocols that must be activated in each of the switches must be specified. In this research, our network topology is based on the ONOS SDN controller and OVS switches, and for connecting the controller to the OVS switches and establishing communication between them, the OpenFlow protocol has been activated on all the switches. To achieve this task, the following commands have been used:

The following commands are used to specify the IP address of the SDN controller, the type of switch, and the activation of the OpenFlow protocol in version 1.3:

```
net = Mininet (  
    topo=NetTopo(),  
    controller=lambda name: RemoteController(name, ip=CONTROLLER_IP,  
    port=CONTROLLER_PORT),  
    switch=partial (OVSSwitch, protocols='OpenFlow13')  
)
```

The following command performs network deployment:

```
net.start()
```

Before executing the script, the ONOS must be run as the SDN controller. To run the ONOS controller, using the CLI command and in the ONOS_ROOT folder, the following command must be executed:

```
bazel run onos-local -- clean debug
```

After successfully running launching the ONOS, for connecting the OVS switches defined in mininet with ONOS, the OpenFlow protocol must be activated (By default, this application is deactivated in ONOS). The following commands activate the *OpenFlow and Forwarding* apps:

```
> app activate org.onosproject.fwd
```

```
> app activate org.onosproject.openflow
```

```
gwdguser@gwdguser-machine: ~
File Edit View Search Terminal Help

gwdguser@root > Connection to localhost closed by remote host. 23:57:29
gwdguser@gwdguser-machine:~$ onos/tools/test/bin/onos localhost
Welcome to Open Network Operating System (ONOS)!

  ONOS

Documentation: wiki.onosproject.org
Tutorials:    tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'logout' to exit ONOS session.

gwdguser@root > app activate org.onosproject.fwd 18:48:48
Activated org.onosproject.fwd
gwdguser@root > app activate org.onosproject.openflow 18:51:50
Activated org.onosproject.openflow
gwdguser@root > 18:52:43
```

Figure 6. 2: Activate the OpenFlow and Forwarding apps in ONOS.

After executing these commands, it can be verified in the ONOS CLI that the OpenFlow protocol is already active, as can be seen in Figure 6.2.

Once the previous command has been completed, in a new CLI terminal in the folder where the developed script resides and execute the following command, "NetTopoScript.py" is the name of the developed script:

sudo python NetTopoScript.py

An example of the script execution is shown in figure 6.3. The ONOS controller provides the overall view of the whole network, and it depicts how all the devices and their connections are added, and the network topology is ready for generating network traffic.

It has not been considered to add a capture of the complete output of the execution of the script in figure 6.3 due to the great extent of the displayed information when generating the traffic between the hosts on the network.

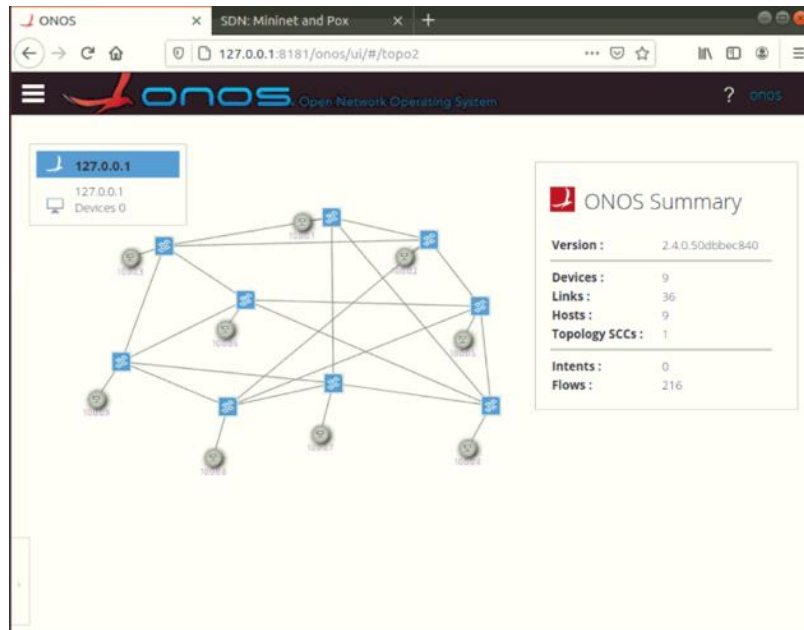


Figure 6. 3: Network Setup from Controller Point of View

For the experimentation, we launch a DDoS flooding attack to evaluate the performance of the BFDD-S framework.

6.3 BFDD-S Framework Operation

The figures in the following present the operation of the proposed framework. In the beginning, we execute a ping command for sending the ICMP packet between two hosts to show the network connectivity between hosts and make sure that the hosts are reachable from each other (Figure 6.4).

The screenshot shows two terminal windows. The left window, titled 'Sending ICMP Packets via Ping Command', shows the output of a ping command from a host at 10.0.0.2 to a host at 10.0.0.2. The output shows 24 successful ping attempts with varying response times. The right window, titled '"Node: h11"', shows the output of a flood ping command from a host at 10.0.0.2 to a host at 10.0.0.2. The output shows that the flood ping failed, with 218398 packets transmitted and 0 packets received, resulting in 100% packet loss.

```

Sending ICMP Packets via Ping Command
gwdg-user@mininetvm: ~
64 bytes from 10.0.0.2: icmp_seq=510 ttl=64 time=0.071 ms
64 bytes from 10.0.0.2: icmp_seq=511 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=512 ttl=64 time=0.087 ms
64 bytes from 10.0.0.2: icmp_seq=513 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=514 ttl=64 time=0.092 ms
64 bytes from 10.0.0.2: icmp_seq=515 ttl=64 time=0.104 ms
64 bytes from 10.0.0.2: icmp_seq=516 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=517 ttl=64 time=0.106 ms
64 bytes from 10.0.0.2: icmp_seq=518 ttl=64 time=0.102 ms
64 bytes from 10.0.0.2: icmp_seq=519 ttl=64 time=0.056 ms
64 bytes from 10.0.0.2: icmp_seq=520 ttl=64 time=0.081 ms
64 bytes from 10.0.0.2: icmp_seq=521 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=522 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=523 ttl=64 time=0.080 ms
64 bytes from 10.0.0.2: icmp_seq=524 ttl=64 time=0.078 ms
^C
--- 10.0.0.2 ping statistics ---
524 packets transmitted, 524 received, 0% packet loss, time 535549ms
rtt min/avg/max/mdev = 0.037/0.078/0.613/0.031 ms
mininet> h11 ping h12
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C

"Node: h11"
root@mininetvm:/home/gwdg-user# sudo hping3 -V -c 100000 -d 9000 -S -w 64 --flood 10.0.0.2
using h11-eth0, addr: 10.0.0.1, MTU: 1500
HPING 10.0.0.2 (h11-eth0 10.0.0.2): S set, 40 headers + 9000 data bytes
auto-activate fragmentation, fragments size: 1480
hping in flood mode, no replies will be shown
^[[AC
--- 10.0.0.2 hping statistic ---
218398 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@mininetvm:/home/gwdg-user#
    
```

Figure 6. 4: Ping command before launching flooding attack.

As mentioned earlier, the new traffic sends to the controller, and the controller transfers the incoming new traffic to the big data pipeline for data processing and attack detection phase; as figure 6.5 shows, the anomaly detection module considers the traffic as normal; therefore, no action required and the packet will be forwarded to the destination port.

```

gwdguser@appserver: ~/spark-2.3.0-bin-hadoop2.7
File Edit View Search Terminal Help
org.spark-project.spark#unused;1.0.0 from central in [default]
org.xerial.snappy#snappy-java;1.1.2.6 from central in [default]
-----
|               |               |               |               |               |
|      conf      |      number  |      modules  |      artifacts  |
|               |      search|dnwlded|evicted||      number|dnwlded| |
|---|---|---|---|---|---|---|---|
|      default  |      12     |      0       |      0         |      0       ||      12     |      0     |
|-----|-----|-----|-----|-----|
:: retrieving :: org.apache.spark#spark-submit-parent
  confs: [default]
  0 artifacts copied, 12 already retrieved (0kB/23ms)
row = {'srcAddr': u'10.0.0.1', 'uniqDstPorts': 1, 'uniqDstIPs': 1, 'sumOfBytes': 31069120}
Normal Flow. No further Action Required.
Anomaly Detected.
Send to ONOS: Need to block 10.0.0.1
urlToPost = http://192.168.0.166:8181/onos/v1/flows/of:0000000000000001?appId=org.onosproject.fwd
response is <Response [201]>

row = {'srcAddr': u'10.0.0.2', 'uniqDstPorts': 98, 'uniqDstIPs': 1, 'sumOfBytes': 9504}
Normal Flow. No further Action Required.
Anomaly Detected.
Send to ONOS: Need to block 10.0.0.2
urlToPost = http://192.168.0.166:8181/onos/v1/flows/of:0000000000000001?appId=org.onosproject.fwd
response is <Response [201]>

```

Figure 6. 5: Legitimate Traffic Detection Message

To generate a DDoS flooding attack, we use hping3 to employ a TCP SYN flood attack, the sample command which we use is as follows:

sudo hping3 -V -c 100000 -d 9000 -S--flood [IP- address]

using this command, we send 100000 SYN packets with a data packet size of 9000 bytes and as fast as possible without waiting for the reply message. During the analyzing the incoming network traffic, if the anomaly detection module identifies any anomaly, it generates a REST request and sends it to the controller to block the attacker as can be seen in the following figure 6.6.

```

gwdguser@appserver: ~/spark-2.3.0-bin-hadoop2.7
File Edit View Search Terminal Help
org.spark-project.spark#unused;1.0.0 from central in [default]
org.xerial.snappy#snappy-java;1.1.2.6 from central in [default]
-----
|           |           |           |           |           |           |           | |
|   conf    | number | modules | search|dwnlded|evicted|| artifacts |
|           |         |         |         |         |         ||         |
|   default |    12  |    0    |    0    |    0    ||    12    |
|           |         |         |         |         |         ||         |
-----
:: retrieving :: org.apache.spark#spark-submit-parent
confs: [default]
0 artifacts copied, 12 already retrieved (0kB/23ms)
row = {'srcAddr': u'10.0.0.1', 'uniqDstPorts': 1, 'uniqDstIPs': 1, 'sumOfBytes': 31069120}
Normal Flow. No further Action Required.
Anomaly Detected.
Send to ONOS: Need to block 10.0.0.1
urlToPost = http://192.168.0.166:8181/onos/v1/flows/of:000000000000001?appId=org.onosproject.fwd
response is <Response [201]>

row = {'srcAddr': u'10.0.0.2', 'uniqDstPorts': 98, 'uniqDstIPs': 1, 'sumOfBytes': 9504}
Normal Flow. No further Action Required.
Anomaly Detected.
Send to ONOS: Need to block 10.0.0.2
urlToPost = http://192.168.0.166:8181/onos/v1/flows/of:000000000000001?appId=org.onosproject.fwd
response is <Response [201]>

```

Figure 6. 6: Anomaly Detection Message by the Framework

After receiving the request from the attack detection module, the controller makes action and creates a new rule to block the attacker’s host. As Figure 6.7 indicates after launching the attack, the host is blocked and the normal ICMP packet sending does not work via the PING command.

```

gwdg-user@mininetvm: ~
Launch DDoS attack "Node: h11"
64 bytes from 10.0.0.2: icmp_seq=510 ttl=64 time=0.071 ms
64 bytes from 10.0.0.2: icmp_seq=511 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=512 ttl=64 time=0.087 ms
64 bytes from 10.0.0.2: icmp_seq=513 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=514 ttl=64 time=0.092 ms
64 bytes from 10.0.0.2: icmp_seq=515 ttl=64 time=0.104 ms
64 bytes from 10.0.0.2: icmp_seq=516 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=517 ttl=64 time=0.106 ms
64 bytes from 10.0.0.2: icmp_seq=518 ttl=64 time=0.102 ms
64 bytes from 10.0.0.2: icmp_seq=519 ttl=64 time=0.056 ms
64 bytes from 10.0.0.2: icmp_seq=520 ttl=64 time=0.081 ms
64 bytes from 10.0.0.2: icmp_seq=521 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=522 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=523 ttl=64 time=0.080 ms
64 bytes from 10.0.0.2: icmp_seq=524 ttl=64 time=0.078 ms
^C
--- 10.0.0.2 ping statistics ---
524 packets transmitted, 524 received, 0% packet loss, time 535549ms
rtt min/avg/max/ndev = 0.037/0.078/0.613/0.031 ms
mininet> h11 ping h12
SDN Controller Blocked the Attacker
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C

root@mininetvm:/home/gwdg-user# sudo hping3 -V -c 100000 -d 9000 -S -w 64 --flood 10.0.0.2
using hui-eth0, addr: 10.0.0.1, htu: 1000
HPING 10.0.0.2 (h11-eth0 10.0.0.2): S set, 40 headers + 9000 data bytes
auto-activate fragmentation, fragments size: 1480
hping in flood mode, no replies will be shown
^[[AC
--- 10.0.0.2 hping statistic ---
2183998 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@mininetvm:/home/gwdg-user#

```

Figure 6. 7: Ping command After Blocking the Attacker.

6.4 Performance Evaluation of the BFDD-S Framework

To evaluate the performance of the framework, we choose the following metrics: CPU and memory consumption of the ONOS controller, average response to the legitimate traffic during the attack, and average time to detect and mitigate a DDoS attack. We use the psutil python library and the htop tool to measure the above-mentioned metrics. The figures 6.8, 6.9, and 6.10 depict the performance metrics under the DDoS attack. The evaluation results clearly show that increasing the number of packets has not so much effect on the performance of the ONOS Controller. Analyzing the CPU and memory consumption, figures 6.8 and 6.9 indicate that in the time that the number of attacking packets is increased under launching a flooding attack, the consumption fluctuation does not increase drastically and it shows that the DDoS attack could not saturate drastically the controller's CPU and memory capacity of the ONOS controller. With the highest number of attacking packets, the controller reaches less than 20% of average CPU consumption and in addition, the average Memory consumption reaches less than 35%.

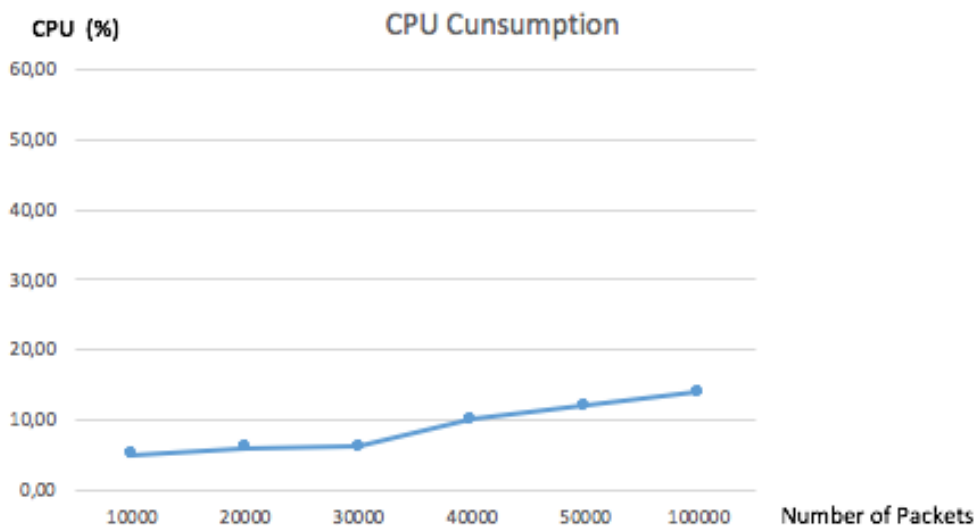


Figure 6. 8: Average Controller CPU Consumption.

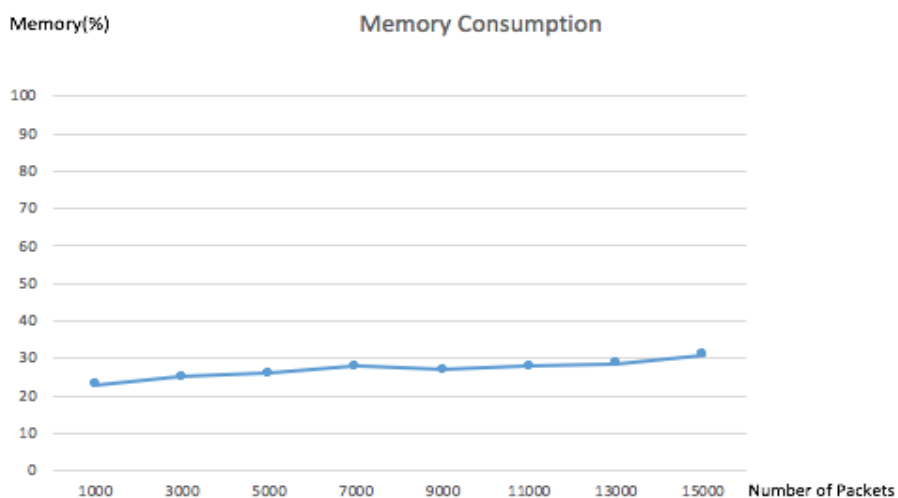


Figure 6. 9: Average Controller Memory Consumption.

On the other hand, figure 6.10 illustrates the average response time by the controller to the legitimate traffic during the attack. It shows by using the BFDD-S framework, there is not so much response delay to the legitimate traffic and the average latency value is not significant and the flooding attack does not considerably affect the processing of the normal traffic.

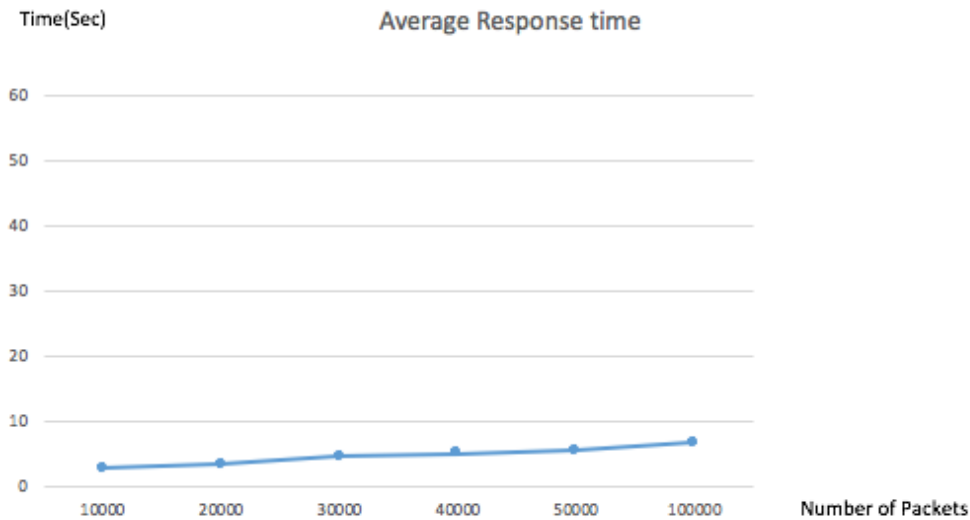


Figure 6. 10: Average Controller Response time to Legitimate Traffic.

In addition to previous analysis, and to thoroughly evaluate the BFDD-S framework and ascertain its performance, we conducted a comprehensive analysis incorporating various methodologies. Our objective was to assess the effectiveness of the framework by comparing it to both SDN-based and traditional approaches employed in countering DDoS attacks. For the first approach, we employed a comparative analysis with an SDN controller using centralized attack detection method, for this purpose the OPERETTA SDN controller was utilized. In fact, OPERETTA, an evolved version of the Pox controller, was specifically designed to detect and mitigate DDoS SYN flooding attacks. By benchmarking the BFDD-S framework against OPERETTA, we could gain valuable insights into its efficacy and potential advantages.

In addition to the SDN-based approach, we also evaluated the BFDD-S framework against traditional technique that are still prevalent in combating DDoS attacks. This technique included the utilization of Firewalls, which serve as a first line of defense. By examining the performance of the BFDD-S framework against the different approach, we aimed to provide a comprehensive understanding of its capabilities and strengths in addressing the evolving landscape of DDoS threats.

In order to evaluate the performance of the BFDD-S framework, we took into account several crucial metrics. These metrics encompassed the CPU and memory consumption of the Controller, the average response time to legitimate traffic during an attack, and the average time required for mitigating a DDoS attack by the controller. By analyzing these metrics, we aimed to acquire valuable insights into the efficiency and effectiveness of the BFDD-S framework when confronted with DDoS attacks. This comprehensive evaluation enabled us to assess the framework's ability to handle such attacks and provided a clearer understanding of its operational performance.

Additionally, the evaluation considered the framework's adaptability and scalability in dynamic network environments. We examined how well BFDD-S coped with increasing traffic loads while maintaining its responsiveness and ability to prevent DDoS attacks effectively.

6.4.1 Performance Evaluation of the BFDD-S Framework with Centralized Detection Method

In first method, to compare our proposed BFDD-S framework with a centralized method, we address one of the primary challenges associated with centralized defense methods: the delay caused by the validation process of incoming packets, which subsequently affects the controller's response time to legitimate flows. This analysis focuses on highlighting the advantages of the BFDD-S method in reducing response time for authorized requests and expediting attack mitigation compared to centralized attack detection methods. In the conducted experiments, we subjected both the BFDD-S framework and OPERETTA to a series of simulated DDoS attacks under controlled conditions. By carefully monitoring the response time of each method, we observed a significant difference in their ability to handle legitimate traffic during attack scenarios.

Evaluating response time to legitimate flows:

The BFDD-S method excels in its ability to detect attacks without introducing delays in the controller's packet handling. Because in this method, all packets are directed to a separate attack detection module for validation, relieving the controller from the responsibility of validating and responding to these requests. Consequently, the time required to respond to authorized requests is significantly reduced. To support this claim, we conducted an experiment to evaluate the response time of the BFDD-S method.

For this experiment, we subjected the network to DDoS flood attacks at varying rates. We then sent requests to a web service and measured the time taken to establish of a complete HTTP session. These results were then compared with the output of the OPERETTA controller. The findings as shown in figure 6.11 explicitly demonstrate that our proposed framework method significantly reduces response time to legitimate traffic when compared to OPERETTA. In instances of low-rate attacks, the OPERETTA controller exhibited response times of 2 to 4 seconds, whereas our proposed method consistently achieved response times below 2 seconds. Moreover, as the attack rate increased, the BFDD-S method exhibited even greater advantages in terms of response time to legitimate flows.

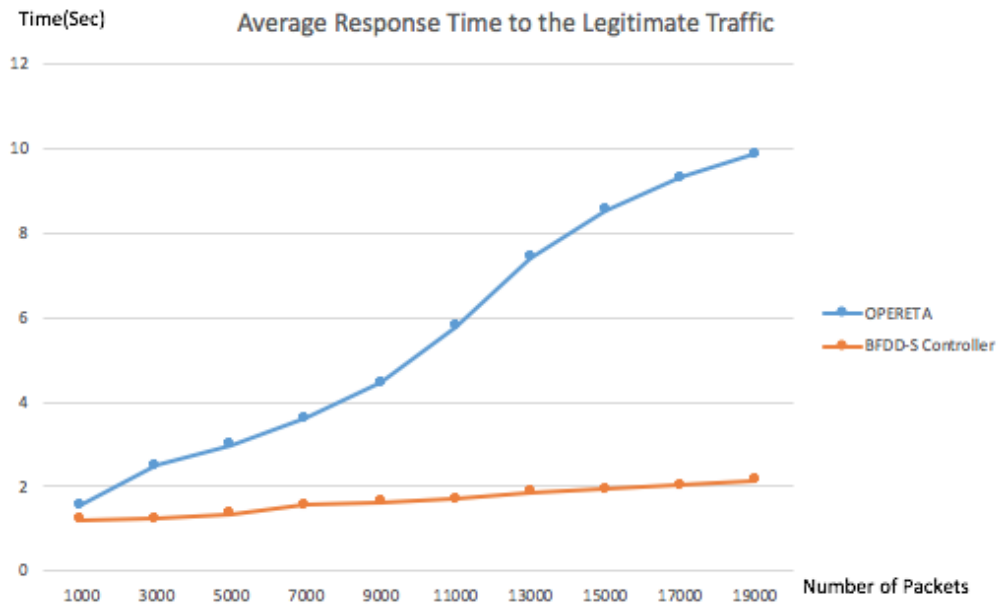


Figure 6. 11: Comparison between BFDD-S and centralized method in terms of average response time to the legitimate traffic.

Furthermore, the experiment evaluated the scalability of the BFDD-S method by subjecting the network to a high-intensity DDoS attack. Under this extreme condition, the BFDD-S framework maintained its efficiency in handling legitimate traffic, with response times remaining well below the tolerable threshold. In contrast, the OPERETTA controller experienced significant delays and struggled to respond promptly during the intense attack.

These results confirm that the BFDD-S method not only excels in its responsiveness to legitimate requests but also exhibits robust scalability, making it a promising solution for defending against DDoS attacks in modern, high-traffic network environments. Its ability to maintain optimal performance even under extreme conditions further emphasizes the practicality and effectiveness of the BFDD-S framework as a comprehensive DDoS defense strategy.

Evaluating Mitigation Time by the controller:

In addition to reducing response time to legitimate flows, we also assessed the time required for the controller to mitigate attacks. To perform this analysis, we subjected the network to a flooding of SYN packets and measured the time required for the controller to mitigate the attacks. Our evaluation (figure 6.12) revealed that the BFDD-S method surpasses centralized methods by providing quicker attack mitigation. In centralized approaches, the controller must bear the burden of detecting and mitigating the attack while also control, maintain and managing the network, resulting in longer mitigation times. The evaluation results, illustrated in the graph, clearly demonstrate the time advantages offered by the BFDD-S method across different attack rates.

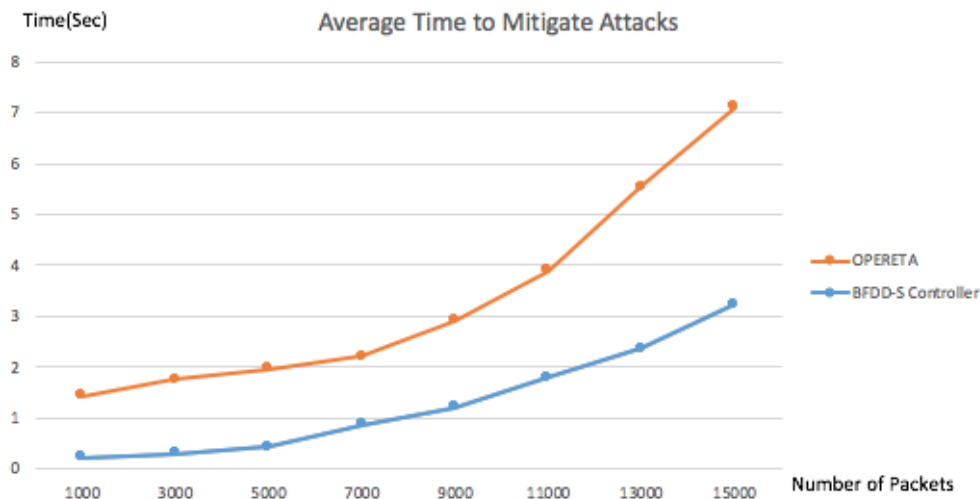


Figure 6. 12: Comparison between BFDD-S and centralized method in terms time required to mitigate DDoS attack by the controller.

Evaluation of Processing Load in the Controller:

To evaluate the processing load of the controller in the BFDD-S method, we conducted a comprehensive analysis by subjecting the simulation network to a flood attack consisting of SYN packets at varying rates. The results in figure 6.13 clearly demonstrate that the BFDD-S method exhibits a significantly lower processing load compared to OPERETTA when the attack rate remains below 9000 packets. This finding highlights the efficiency and effectiveness of the BFDD-S method in efficiently handling and managing the processing load during moderate-intensity DDoS attacks. However, as the attack rate increases beyond 9000 packets, the processing load of the BFDD-S method approaches that of the OPERETTA controller. This observation suggests the need for further optimization or scaling strategies to ensure optimal performance of the BFDD-S method under high-intensity DDoS attacks.

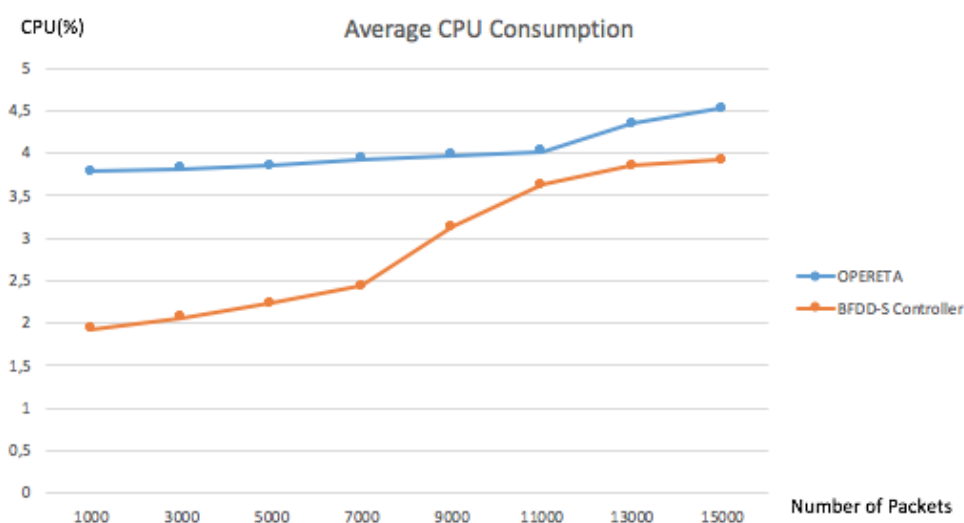


Figure 6. 13: Comparison between BFDD-S and centralized method in terms of processing load on the controller during the DDoS attack.

Evaluation of Controller Memory Consumption:

Furthermore, we evaluated the memory consumption of the controller in both the BFDD-S method and the centralized approach. The results in figure 6.14 indicate that the proposed framework exhibits slightly lower memory consumption values compared to the centralized method. This implies that the BFDD-S method is more resource-efficient and capable of effectively managing memory utilization during DDoS flooding attacks. However, it is worth noting that as the attack rate varies, the memory consumption of the BFDD-S method approaches that of the centralized method. This finding underscores the need for continuous monitoring and optimization to ensure efficient memory utilization, particularly under high-intensity attack scenarios.

Overall, the evaluations of processing load and memory consumption reveal the strengths of the BFDD-S method in mitigating DDoS attacks with reduced resource utilization. These insights serve as a basis for further optimization and enhancement of the framework to enhance its scalability and performance in real-world DDoS attack scenarios.

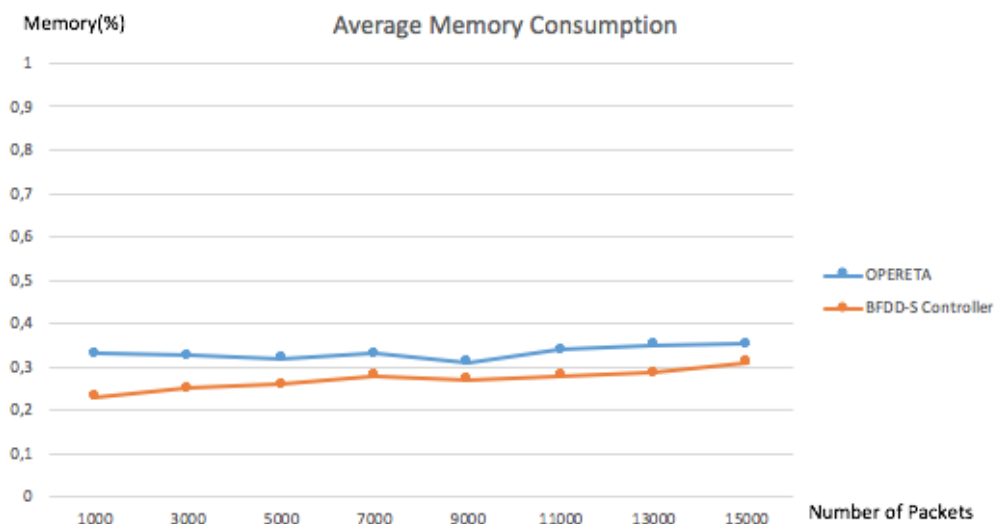


Figure 6. 14: Comparison between BFDD-S and centralized method in terms of memory consumption during the DDoS attack.

In conclusion, the comparison between the BFDD-S framework and OPERETTA provides compelling evidence of the proposed framework's capability to effectively reducing response time for authorized requests, expediting attack mitigation and enhance the overall resilience of the network compared to centralized defense methods. By offloading the validation process to a separate detection module, the SDN controller is relieved of a heavyweight traffic, leading to reduced response delays and improved network security. These experimental findings underscore the potential of BFDD-S as a promising solution for bolstering network security and defending against DDoS attacks in modern, dynamic environments.

6.4.2 Performance Evaluation of the BFDD-S Framework with Traditional Methods

In this section, we compare the proposed defense method with a common traditional method, which is a firewall. Firewalls have long been employed as a fundamental security measure to protect networks from unauthorized access and potential threats. In our comparison, we specifically use iptables, a flexible command-line firewall tool that comes pre-installed on most Linux distributions. By conducting this comparison, we aim to assess the effectiveness and limitations of the traditional firewall approach when faced with TCP-based DDoS attacks, as opposed to the proposed defense method. Through a series of carefully crafted experiments, we evaluate how each approach handles varying attack intensities and the extent to which they can successfully mitigate the impact of such attacks on the network.

To compare the performance of the framework with traditional methods, again we choose the following metrics: CPU consumption of the ONOS controller, and average response time to the legitimate traffic during the attack for both the proposed framework and the traditional method. To measure these metrics, we use the psutil python library and htop tool for evaluations, such as response time to authorized users and the amount of processing load imposed on the server. Figures 6.11, and 6.12 depict the performance of these metrics under the DDoS attack for both methods. The evaluation results clearly show that increasing the number of packets has not so much effect on the performance of the ONOS Controller. On the other hand, as can be seen in figures the firewall method imposes a large amount of processing load on the server's CPU when dealing with attacks.

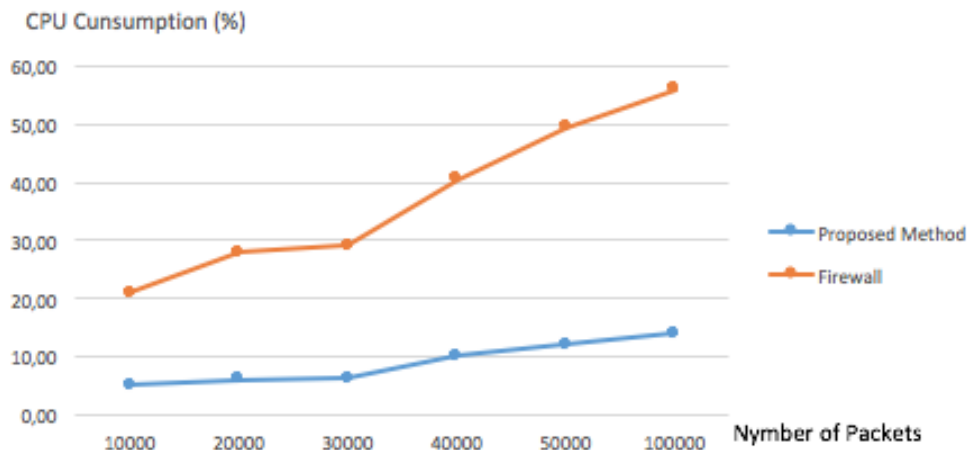


Figure 6. 15: The average processing load in the traditional and proposed method.

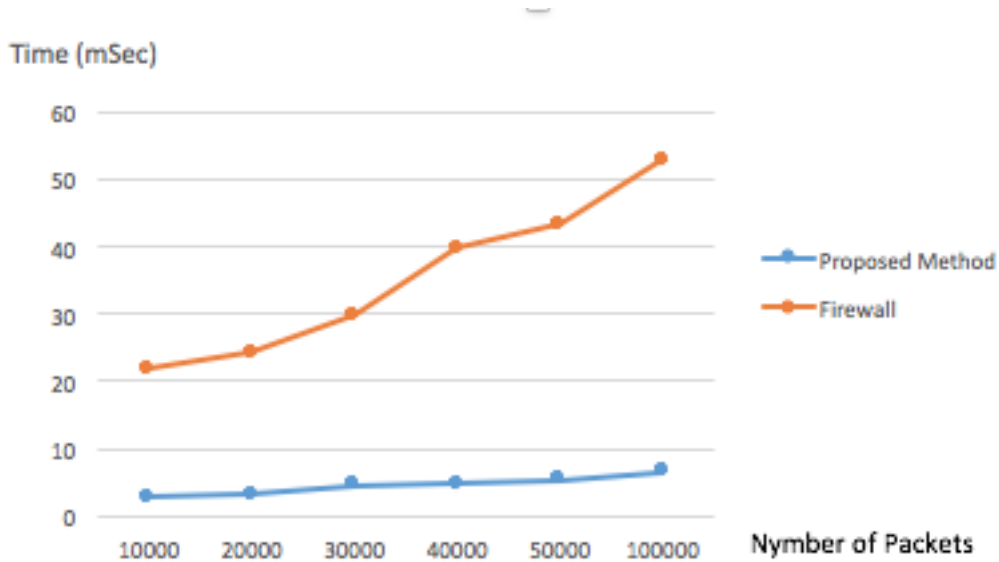


Figure 6. 16: The average response time to the legitimate flows in the traditional defense and proposed methods.

Analyzing the CPU consumption indicate that the moment the number of attacking packets is increased, in both cases the CPU consumption does not increase drastically and it shows that the DDoS attack could not saturate the controller’s CPU and memory capacity of the ONOS controller. The controller with the highest number of attacking packets reaches less than 20% of average CPU consumption, as can be seen in figure 6.11, the firewall method imposes a large amount of processing load on the server’s CPU when dealing with attacks. As the attack rate increases, so does the CPU consumption. To measure the response time to the legitimate traffic by the server for normal users in the traditional and proposed method, we performed an experiment by generating several different traffic rates consisting of SYN packets on the webserver. Figure 6.12, illustrates the average response time by the controller to legitimate traffic during the attack. It shows by using the proposed framework, there is not so much response delay to the legitimate traffic and the average latency value is not significant and the flooding attack does not considerably affect the processing of the normal traffic. In the traditional countermeasures method, with increasing attack rates, a significant amount of time is spent responding to unauthorized requests, and the response time to authorized users’ requests increases dramatically.

6.5 Chapter Summary

In this chapter, for evaluating the performance of the framework, first, we design and implement a simple leaf-spine network architecture physical testbed by using four Dell Power Edge R430 servers and four bare metal EdgeCore AS 4610 switches that support OpenFlow protocol and one HP2530-8G as a management switch.

For emulating an SDN network architecture, we use this simple but real test bed as an underlay network, we use Mininet to define the topology on one server, consisting of one Web server and several hosts which are all connected to an SDN-based switch. Then, we generate normal (ICMP packet) and anomaly traffic (TCP SYN flood) simultaneously and walk through our approach, the network traffic captures via the SDN controller and process through the data

pipeline for attack detection, and outcomes are stored in the data store. The CPU and memory consumption of the ONOS controller, the average response to legitimate traffic during the attack, and the average time to detect and mitigate a DDoS attack, are the parameters we consider for our assessment. Moreover, we use the psutil python library and the htop tool to measure the above-mentioned metrics.

Furthermore, to evaluate our proposed framework, its performance and assess its effectiveness, we compare the proposed defense method with an SDN controller using centralized attack detection module(OPERETA) and a common traditional method which is a firewall (Iptables). For this goal, various metrics have been considered, these metrics include: average response time to legitimate traffic as well as the time required to mitigate attack by the controller and CPU and memory consumption of the ONOS controller during attacks. The conclusion of this chapter would clarify the answer to research question number 6, which is:

- **What is the effect of integrating a detection module using the big data pipeline and developing the controller to detect and mitigate DDoS attacks in software-defined networks?**

Chapter 7

7 Projects Contribution and Research Use Cases

During this research, we had the opportunity to involve in three European projects which can be considered research use cases. In each of these projects, we contribute to various work packages by offering the result of different parts of this research. In this section, these projects are introduced and described briefly how the SDN paradigm can offer elasticity and cope with some specific key network issues and additionally how it can provide a possibility to modify the network infrastructure on demand to match the needs of any organization by incorporating software controls and automation into practically any LAN or WAN management system (Table 7.1 describes briefly these projects and our contribution).

Table 7. 1: Projects Contributions

SDN USE CASES	Duration	Project Funding	Our Research Contribution
NEPHELE Project	01.02.2015 - 31.01.2018	European Union, Horizon 2020	The result of our evaluation of various machine learning algorithms to detect DDoS attack is used to enhance the security of the agent which is responsible to provide communication between the control plane and the new hybrid optical-electrical switches.
SENDATE Project	01.04.2016 - 31.03.2019	German Federal Ministry of Research and Education(BMBF)	As a part of the network orchestration of distributed data center structures, our research solution is proposed in this project to make data center interconnectivity secure and reliable.
AI-NET-PROTECT Project	01.02.2021 - 31.01.2024	German Federal Ministry of Research and Education(BMBF)	Our research framework is proposed in this project as a solution for work package four which is Strong Integrated Security & Trust for enhancing and improving current machine learning/artificial intelligence systems.

7.1 Nephele Project.



The first project was the Nephele Project. The purpose of this project, according to its description, was to provide an “end-to-end scalable and dynamically reconfigurable optical Architecture for application-aware SDN Cloud Datacenters (NEPHELE)” [195].

7.1.1 Project goals

The project intended to construct a dynamic optical network architecture capable of increasing data center efficiency and hence lowering costs and conserving energy. It offers increased scalability in cloud infrastructures by bypassing existing architectural limits. A data center has traditionally been assigned a defined percentage of equipment for processing and storage capacity, among data transmission. The ongoing growth of cloud-based applications involves a massive expansion of the data center's fundamental infrastructure, which from an economic point of view, requires an unsustainable, non-linear development of network components. At the start of the project, the principle was resource disaggregation and SDN concepts. Therefore, it was necessary to construct an upper control layer with a northbound interface to the application layer as well as southbound access to provide hardware abstraction and dynamic network configuration.

The research team is relying on the massive capacities of optical fibers and intends to deploy a network of optical switches to achieve two major objectives: 1) the optimum mixture of high network bandwidth and 2) reducing the cost. By considering the SDN controller as the major component for the development of the project, our task in this project was to emerge a novel control layer, which provides an application-defined network topology and integrates hardware and software virtualization through a hybrid optical infrastructure. This involves the replacement of the control logic part of network equipment with one or additional central software modules. These are responsible for orchestrating the flow of data by the needs of cloud-based applications. To accomplish this goal, the OpenFlow protocol was enhanced and modified to support optical switch needs. Additionally, for the Application Programming Interface, many Representational State Transfer (REST) protocols were considered and studied. Moreover, a module was planned to be included in the developed SDN controller to support virtual machine migration to other data centers in the future. Besides other goals of the NEPHELE project, an initial objective was to provide an extension for the control plane of Software Defined Networking (SDN) for the dynamic allocation of resources of an optical data center network.

To achieve this purpose, an agent can be considered as a part of the control plane unit of an SDN network that facilitates the interaction between a central SDN controller and a non-SDN data plane device. Since the non-SDN devices can be mentioned as the legacy network devices which have not by default any capability to interact and comprehend the OpenFlow protocol. Therefore, the major responsibility of an SDN agent is to interpret the OpenFlow commands from an SDN controller into device particular commands of the underlying network devices and vice versa. Therefore, in the NEPHELE project, we developed an agent, called SDN Optical Agent (SOA), to provide communication between the control plane and the new hybrid optical-electrical switches introduced in the NEPHELE project by other partners. For providing the communication, the agent converts the control plane instructions of an OpenFlow-based controller into the optical-electrical switch configurations.

Since the hybrid optical-electrical switches are based on the FPGA framework, therefore our approach is to develop an agent between the SDN controller, and the FPGA-based switches which converts the OpenFlow messages conducted by the controller to FPGA framework-specific messages. The FPGA framework messages can be then used to program Hybrid Optical Electrical switches and accomplish the flexibility and automation guaranteed by the SDN

network model. Figure 7.1 demonstrates the high-level design of the agent with respect to the SDN controller and FPGA framework.

7.1.2 Our Contribution

The major goal of this project is to design and develop an agent (SOA) to connect optical-electrical switches with an OpenFlow controller and OpenFlow-based switches simultaneously. For enhancing the security of the agent, the result of our research for evaluating different machine learning algorithms for DDoS attack detection is used to implement the appropriate machine learning at the agent level.

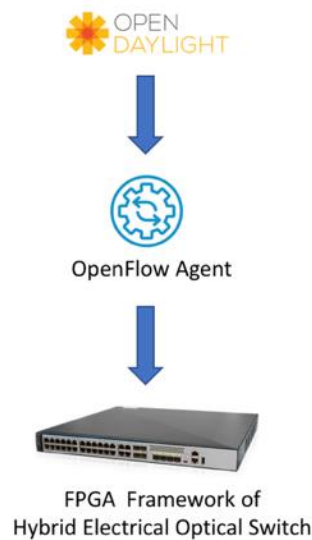


Figure 7. 1: Agent Position in SDN Architecture

7.2 SENDATE Project.



The second project was the SENDATE Project [196], which is the abbreviation of Secure Networking for a Data Center Cloud in Europe (SENDATE). According to the project description, the SENDATE project is a Celtic-Plus cooperative project under the umbrella of EUREKA.

7.2.1 Project Goals

The major motivation behind this project was secure and adaptable data center interconnection. We contributed to this project via the SENDATE-Secure-DCI sub-project, which engages in investigation and development activities concerning network designs as well

as technologies for the network orchestration of distributed data center structures. The major objective of this project is to develop a novel distributed data center, which will be capable of offering more elastic infrastructures, storage space, and protected processing for the clients. We investigated the state-of-the-art data center interconnects during the early stage of the project. This involves collecting information on market needs and application scenarios, investigating data center traffic prototypes, and creating and assessing distributed data center designs. A further goal of this project is to incorporate SDN-based control into distributed data center networks, as well as to achieve virtualization of processing, resource orchestration, and storage supplies. This was accomplished by developing the new SDN controller or using the existing controller together with resource orchestrators. In this project, we were also entailed in the creation of the SDN-based testbed for a distributed data center as part of the technical review process for the ideas and implementations under consideration.

In this project, we can portray our tasks in a nutshell as follows:

- Researching the most recent developments in data center interconnects.
- Information gathering, data center traffic model analysis, and distributed data center design evaluation.
- Combine virtualization of computing and storage resources with SDN-based control to achieve resource orchestration.
- Implementing new network controllers and resource orchestrators.
- Participating in the setup of a testbed for the distributed data center.

In this project, we set up the orchestration framework for distributed data centers, which integrates the SDN controller and cloud computing components. This framework provides fine-grained control of distributed data centers, from the physical machines to the network resources, such that facilitates core concepts in cloud computing and distributed DCs, like multi-tenancy, virtualized function/service chaining, and Simplified interfaces for consolidated administrations and operations.

7.2.1.1 DCI Orchestration Framework and Testbed Setup

The aforementioned orchestrators as well as independent implementation are all optional to our design. On one hand, the idea is to use existing solutions for DC orchestration and lay our focus on the DCI part. On the other hand, it is still open to implementing our orchestrator instead of using Heat, Tacker, XOS, or Kubernetes. This can be a light-weighted orchestrator that simply applies REST API and reduces superfluous functionalities to secure DCI orchestration. The selection criteria are as follows:

- Functions covered,
- Open-source (industry-friendly license),
- Integration of SDN controllers like ODL, ONOS,
- Large community,
- Easily extensible,
- Well documented,
- Compatible with the NFV MANO framework,
- DCI support

Regarding this list especially DCI support, we further have the following considerations. First, distance will very possibly make the network the bottleneck of chaining functions/services. The orchestrator platform which supports the network should gain more focus. Second, the heterogeneity of DCs seems to be more common. A “good” orchestrator should consider this. Third, “dynamic” orchestration should be well supported as it is used for function chain healing and refinement. Our first choices are CORD and OPNFV ONAP [197]. The former is the main focus in SENDATE-Secure-DCI, while the latter is used in the SENDATE Working Group Datacenter as the testbed framework. XOS is the integrated orchestrator of CORD used to implement NFV MANO in the DCI network. In parallel, we also directly call REST API to implement orchestration functionalities in accordance with fundamental control concerns. Since both deploy OpenStack as a core component, Tacker should be a good alternative to do NFV MANO. The traditional DCI components are routers or multilayer switches. The configuration is mostly distributed. The customers are separated by VLAN/VRF. The DCI provides layer 3 and/or layer 2 tunneling between data center edge (DCE) devices. DCI of automated DCs is shown in Figure 7.2. DCEs still connect to fabrics, which, however, are managed by a separate control plane now.

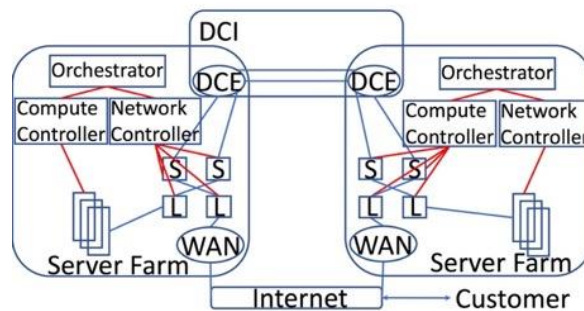


Figure 7. 2: A Simplified DCI Architecture for DCs with Automation. S: spine, L: leaf.

different kinds of controllers in all DCs as well as DCI channels. This orchestrator achieves the united orchestration/control plane. We move the functionalities of the orchestrators within DCs, to the global orchestrator. Another architecture is shown in Figure 7.3 Orchestration

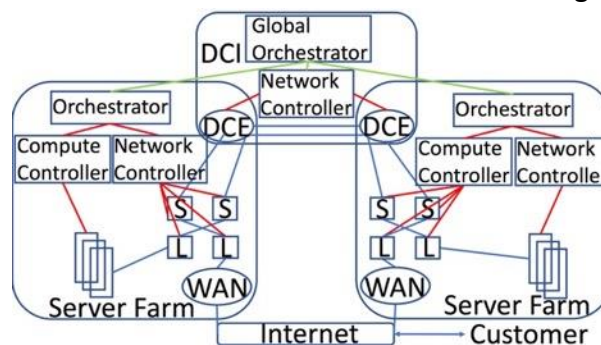


Figure 7. 3: Intermediate Orchestrator or Orchestrator Cascading.

may already exist in distributed DCs. We set up an orchestrator in DCI to play a role as either an “intermediate” or a “commander” of those distributed orchestrators. Compared with the former architecture, this architecture can reduce the administrative load on the global

orchestrator, minimize the changes to DCs, and be easy to scale in case of a large number of DCs.

7.2.2 Our Contribution

In this project, for providing virtualization of processing, resource orchestration, and storage supplies, the SDN-based control is used to incorporate into distributed data center networks. This was accomplished by developing the new SDN controller or using the existing controller together with resource orchestrators. For choosing the proper SDN controller, our research result for evaluating different SDN controllers is used. Therefore, in this project, the ONOS is used as the main SDN controller to provide fine-grained control of distributed data centers, from the physical machines to the network resources, such that facilitates core concepts in cloud computing and distributed DCs, together with the CORD which is used for orchestrator platform to support network, the heterogeneity of DCs, and providing the dynamic orchestration to support for function chain healing and support. Furthermore, we contribute actively to the conceptual investigation, design, and modeling of the DCI prototype among with deploying the Testbed.

The final objective of this project, after successfully implementing the aforementioned goals, is to provide security for the traffic which are transferred in distributed DCs. Currently, the DCs progressively are threatened by DDoS attacks. We offer our research solution to be implemented in each and every SDN-based DCs to detect and pretend DDoS attacks at the controller level and for the DCI channel.

7.3 AI-NET-PROTECT Project.



The third project is the AI-NET-PROTECT [198] Project which is still an ongoing project at the time of writing this thesis. The overall description of the project is offering robust and protected infrastructure networks which are functioning on reliable devices.

7.3.1 Project goals

The major purpose of this project is to quicken Europe's digital conversion via intelligent network automation at different network parts, including the edge, metro, core, and data centers. AI-NET will investigate a variety of use cases covering the technological trials associated with the deployment and operation of services at the network edge, to experience the various scenarios and deployments associated with each use case, and thus the technical needs and associated values. AI-NET will study and implement technologies particular for an edge infrastructure, which is categorized by an enormous number of edge spots, a blend of base methods for virtualization frameworks and transport networks, site configurations, resource-constrained compute environments, and different hardware, and ultimately supporting vital services in modified network parts.

Artificial intelligence (AI) and machine learning (ML) must be used to supplement or replace conventional optimization and prediction techniques to handle the ensuing complexity. Although, the five work packages have been defined for this project, in the following we explain our collaboration in the packages with the approach for enhancing the SDN usage and capabilities.

7.3.2 Work Package Three: AI-based Network control & service automation

In this work package, with the support of OpenFlow agents from the NEPHELE project, we plan to create an automation method for legacy networks. An OpenFlow agent acts as a bridge between an SDN controller and a conventional data plane. Once the OpenFlow commands sent by the controller have arrived and been transcribed, it sends them to the forwarding agents on the data plane. Furthermore, we want to contribute by developing an enhanced monitoring system that will be used for various monitoring applications such as traffic accounting and network security monitoring in real-time. Individual flow data are collected, processed, and stored as part of the proposed system, which is used for the above-mentioned purpose. Figure 7.4 depicts the overall planned contribution in this project.

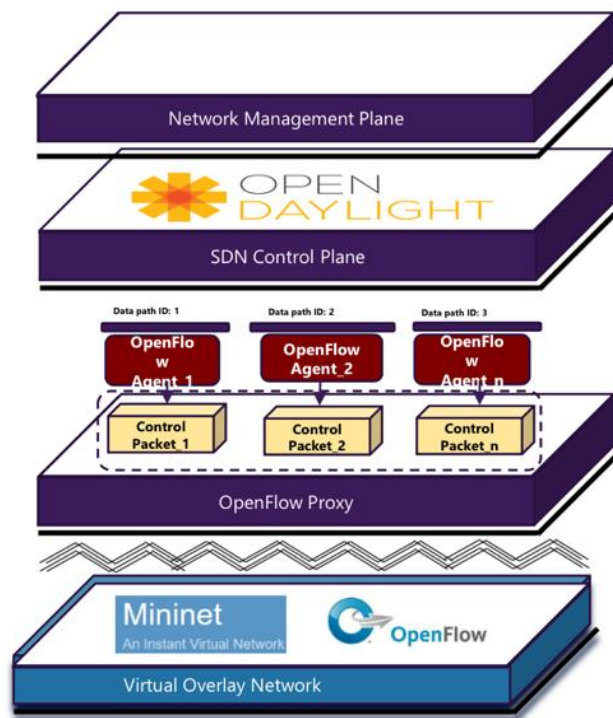


Figure 7. 4: Overview of our Contribution to WP3.

7.3.3 Our Contribution

In this project and terms of security, we see enhancing and improving current machine learning/artificial intelligence systems for detecting/mitigating various types of network threats.

We use the fact that a centralized controller is the optimal location for monitoring network flows in order to identify any anomaly, and therefore, in conjunction with the programmability, the mitigation method will be implemented much more rapidly and effectively. Therefore, instead of implementing the intrusion detector at a few critical locations, the SDN controller can provide the possibility for the security applications to select when and how to monitor the network dynamically.

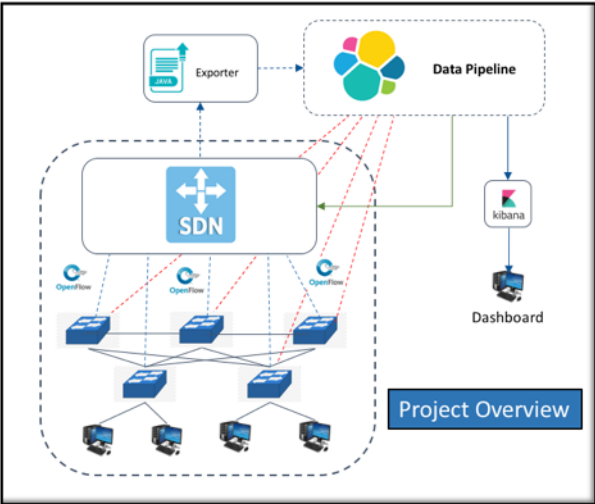


Figure 7. 5: Overview of our Contribution to WP4

As figure 7.5 indicates, this research is initiated as our contribution to this project with some enhancement in architecture such as creating the graphical dashboard and improvement in performance, for instance, defining a methodology for automating the deployment of the overlay network, for managing and configuring systems, scheduling applications, automating and chronizing tasks.

Chapter 8

8 Conclusion and Future Work

In this chapter, we conclude our research by providing a summary and offering an outlook on potential future work.

8.1 Introduction

The SDN architecture is a game-changer in controlling and management of the network as it includes particular features that improve numerous network functions and address intractable issues that exist in traditional networks. By decoupling the control plane from the data plane, the software-defined networking (SDN) architecture offers benefits for both the reprogramming and the centralized management of the network. The network's programmability and centralization of control expedite the creation of new network capabilities through rapid prototyping; In general, the majority of network services present in traditional systems can be implemented in the SDN as simple software implementations. While SDN offers numerous benefits for controlling, maintaining, and implementing network environments, it was initially designed with a lack of security measures. Therefore, similar to the traditional network, the SDN network is exposed to various security risks in its architecture design, where it tries to centralize all the controlling, maintenance, and administration of the network into a single entity. This single centralized unit triggers some new security challenges. One of the most considerable threat factors in SDN structure is the possibility that an attacker attempts to compromise the proper performance of an SDN controller at the control plane level. The centralized nature of SDN, which relies on a single controller, creates a potential point of failure that can be exploited by DDoS attacks, ultimately compromising the entire network.

The main objective of this research is to address SDN security concerns by integrating big data analytics tools and machine learning approaches to provide security defense for the SDN controller. These mechanisms offer fast and reliable malicious activity detection based on monitoring the network flows. Currently, the SDN concepts are mostly used in data center design; therefore, the result of the research is intended to enhance network security in the SDN-based data center environment.

8.2 Conclusions about the Research Questions

This research was motivated by the requirement for a scalable and reliable framework for SDN networks against DDoS attacks. The prime objectives revolve around two approaches: first, separating the detection module from the SDN controller to proactively prevent performance

degradation of the SDN controller, and second, leveraging a big data pipeline to provide a scalable, real-time, and reliable attack detection method.

Since the initiation of this research, some questions have arisen, and they must be addressed through subsequent research and study.

What are the SDN attack vectors?

Compared to traditional networks, security in an SDN network is more complex due to the addition of new components. The central controller can be an attractive target for attackers, and the use of open-source applications and interfaces can make it difficult to define security policies. The SDN architecture was introduced without any security facility in its design. Therefore, mitigating the risk of attack in the SDN network requires a protection-in-design approach to provide suitable defense against different network attacks. To design and implement a robust and resilient attack detection and mitigation method, it is crucial to outline the different SDN attack vectors.

Seven attack vectors can be identified when examining the security vulnerabilities of the SDN architecture based on in-depth research.:

Forged or fake network traffic, network device vulnerability, attacks on the communication channel of the plane of control, SDN controller vulnerabilities, lack of mechanisms to guarantee the trust between the controller and applications, administrative system vulnerabilities, and lack of reliable resources for forensic activities and remediation.

Which machine learning classifier algorithm is suitable for detecting DDoS attacks?

In this research, to improve the accuracy of network traffic classification, we utilize machine learning supremacy in the detection module of our proposed framework. The mission of the detection module is to classify whether the incoming network is normal or classified as an attack based on the features of the incoming traffic.

Therefore, to identify the most appropriate classifier to incorporate into our framework, five prevalent classification algorithms such as Logistic Regression, Naïve Bayes, K-Nearest Neighbor, Decision Tree, and Random Forest have been evaluated by using two well-known datasets (NSL-KDD and UNSW- NB15) for network intrusion detection systems.

Based on the evaluation results, we concluded that the Random Forest classifier offers the best performance in terms of accuracy, precision, and recall compared to the other classification algorithms. Therefore, in this research, we have selected this algorithm to be implemented in the data processing phase of our proposed data pipeline utilizing Apache Spark for the attack detection phase.

How to implement a data pipeline to detect malicious network traffic in the networks?

In this framework, a few big data analytics tools are used to produce a data pipeline infrastructure. For this purpose, we use Apache Kafka for message queuing, Apache Spark for data processing, and Elasticsearch for storing processed data. Additionally, the Spark

Streaming system from the Apache Spark ecosystem is involved in processing data flows, storing them in databases, or reflecting them in a graph to provide a better view for the user.

The features and specifications of each one of these tools are the main reasons behind using them to implement the big data pipeline infrastructure. Specifically, by taking advantage of the short response time, high efficiency, error tolerance, high scalability, and ability to process fast streams of Apache Kafka, together with the real-time stream processing, fault tolerance, scalability, and lightning-fast analytic engine of Apache Spark, and the horizontal scalability, multi-tenancy, and speed of Elasticsearch, we can achieve the major objective of this research, which is to offer a highly scalable, reliable, and almost real-time framework to detect and mitigate DDoS attacks in SDN networks.

How can a detection module be integrated into the SDN controller for the detection and prevention of a DDoS attack in the SDN environment?

When the SDN-based switch sends the incoming traffic to the controller, the attack detection systems are not able to detect abnormal activities from only one packet. therefore, the controller requests the entry of several packets of the same flows into the attack detection system. To facilitate this process, we created a Java API called Extractor within the ONOS core. This API is responsible for collecting statistical information, aggregating flows, and converting the resulting data into a suitable format (such as JSON) for transmission to the detection module. Given that the ONOS controller serves as the basis for our proposed framework, and considering that ONOS employs the OSGi framework with Karaf to divide applications into bundles, we followed these steps to develop the Extractor API:

- Creating an API directory template
- Registering the API with Karaf
- Importing services from other modules
- Compiling the API and plugging in

What is the effect of integrating a detection module using the big data pipeline and developed controller to detect and mitigate DDoS attacks in software-defined networks?

To evaluate the performance of the framework, we choose the following metrics: CPU and memory consumption of the ONOS controller, average response to legitimate traffic during an attack, and average time to detect and mitigate a DDoS attack. Furthermore, we compare the proposed defense method with a common traditional method which is a firewall (Iptables). Next, we conducted a test scenario in which we simultaneously generated normal (ICMP packet) and anomalous (TCP flood) traffic. During this experiment, we applied our proposed approach to capture network traffic via the SDN controller and process it through the data pipeline to detect attacks, with the resulting outcomes stored in the data store.

The evaluation results indicate that increasing the number of packets had a relatively minor impact on the performance of the ONOS Controller. Based on the analysis of CPU and memory consumption, our results indicate that increasing the number of attacking packets during a flooding attack did not significantly increase consumption fluctuations. This suggests that the DDoS attack was unable to drastically saturate the ONOS controller's CPU and memory capacity. Moreover, the controller can handle legitimate traffic with a reasonable delay time.

8.3 Research Summary

In this research, we present an intrusion detection system using distributed processing technologies based on the combination of machine learning algorithms and big data pipeline infrastructure in SDN-based networks for proactively preventing the performance degradation of the SDN controller. This proposed framework is called **BFDD-S** (Big data Framework for DDoS attack Detection in SDN networks). To identify the classifier model with the best classification accuracy for employing a machine learning algorithm using Spark, we evaluated the effectiveness of several classification techniques.

The first two chapters introduce the research and discuss the motivation behind it, as well as summarize related work. Chapter three is used to describe the essential associated background and correlated information. In chapter 4, we evaluate different SDN controllers to distinguish the appropriate controller to be utilized in our research. There exist several open-source controllers in the research and academic domain. This chapter considers and evaluates the five most popular controllers, such as NOX/POX, Ryo, Floodlight, ODL, and ONOS. We classify the SDN controllers based on the following characteristics: Architecture, Interfaces (South-, Northbound), Scalability, Modularity and Extensibility, and Performance. Our evaluation result indicates that the ONOS controller has the capabilities and features which is vital for our research; Therefore, in our proposed solution, we choose ONOS as the SDN controller and implement it in our framework.

In chapter 5, we propose our solution against DDoS threats in SDN-based networks. The framework, which is called BFDD-S, is a heterogeneous system. In other words, in this method, the detection phase will be done outside of the SDN controller, and the information gathering and mitigation phase will be done by the controller. To provide scalability, real-time, and fast processing, we have designed and implemented a data pipeline infrastructure using various big data tools. For implementing the data pipeline infrastructure, we use Apache Kafka for message queuing, Apache Spark for real-time data streaming and machine learning processing, and Elasticsearch for storing data. We use the MLLib API library as a part of the Apache Spark ecosystem for employing a machine learning algorithm.

For evaluating the effectiveness of various machine learning algorithms for intrusion detection, we have applied five well-known general-purposed supervised machine learning algorithms, such as Logistic Regression, Naïve Bayes, K-Nearest Neighbor, Decision Tree, and Random Forest, using the two common intrusion detection datasets, NSL-KDD and UNSW-NB15. In our investigation, three major metrics, Accuracy, Precision, and Recall, were employed to determine which algorithm should be implemented in the DDoS detection module using Apache Spark in our proposed framework (BFDD-S).

The result based on the NSL-KDD dataset presents the Decision Tree and Random Forest gives the best accuracy, which is more than 98%, followed by the Logistic Regression algorithm which offers more than 89% accuracy. This classification report depicts that the Naïve Bayes has the weakest performance, with a score of 88% accuracy. Although the two classifiers delivered more than 98% accuracy, the table indicates that Random Forest has the best result. In addition, the results based on the UNSW-NB15 dataset indicate that the Random Forest algorithm has the highest accuracy of 99%, which is significantly better than other classification algorithms such as the Decision Tree, K-Nearest Neighbor, Logistic Regression,

and Naïve Bayes algorithms.

Moreover, the classification report based on the UNSW-NB15 dataset shows that the Naïve Bayes algorithm achieved high accuracy (>88%); however, it also yielded low scores in recall and precision (>64% and >78%, respectively), indicating that Naïve Bayes had the weakest performance compared to other classification algorithms utilized in this research. Based on the evaluation results, we concluded that the Random Forest classifier offers the best performance in terms of accuracy compared to the other classification algorithms. Therefore, in this research, we have chosen this algorithm to be implemented in the data processing phase of our proposed data pipeline using Apache Spark for attack detection.

According to the outcome of the evaluation, we chose the Random Forest classifier to be employed in Apache Spark as the data processing part of our data pipeline infrastructure for accurate and rapid anomaly detection.

To evaluate the proposed framework, we simulated an SDN-based environment using a simple testbed with real hardware and leaf-spine architecture. We utilized ONOS as the controller and Mininet to define and emulate the SDN network topology, which consisted of several hosts connected to the SDN controller. We generate the normal ICMP traffic using the Ping command and DDoS traffic using the hping tool to assess the response of the proposed framework. We consider the CPU, memory consumption, and average response time to legitimate traffic as evaluation metrics. Analyzing the CPU and memory consumption indicates that during performing the DDoS attack, with the highest number of attacking packets, the controller reaches less than 20% of the average CPU consumption. In addition, the average memory consumption reaches less than 35%, which indicates that in both cases the consumption fluctuation does not increase drastically, and it shows that the DDoS attack could not saturate the controller's CPU and memory capacity of the SDN controller.

The results show that during the DDoS attack, the average response time to legitimate traffic was less than 5 ms when using the BFDD-S framework. Even at the highest number of attacking packets, the controller still responded to normal traffic in less than 10 ms. This suggests that the framework did not cause significant delays in processing normal traffic and the DDoS flooding attack did not greatly affect the controller's ability to handle normal traffic. In this research, we propose to create a data pipeline that integrates machine learning techniques and big data analytics tools to develop a fast, reliable, and scalable security framework for SDN-based network environments to detect and mitigate DDoS attacks. The proposed framework will incorporate features such as high scalability, the ability to process fast streams of events using Kafka, real-time stream processing, the scalability of Apache Spark, and the horizontal scalability and speed of Elasticsearch. This framework will aim to enable early detection and mitigation of DDoS attacks in SDN-based network environments.

These results imply that the BFDD-S framework can provide a scalable and reliable platform to prevent DDoS attacks in SDN networks in comparison to other traditional methods. Another major advantage of this solution compared to the other solution is that most of the related methods do not consider southbound saturation during DDoS attacks. If the bandwidth of the southbound interface in the SDN network is saturated by heavy traffic, the network flows will not reach the controller, and these methods cannot perform their proposed detection methods. However, in the BFDD-S framework, if the southbound is saturated with heavy

traffic during a DDoS attack, we can configure the network devices to send the network information via common network protocols such as Netflow and IPFix to the data processing pipeline, and the process will be the same, and if any anomaly activity is detected, the Apache Spark will generate and send a RestAPI request to the controller to block the attacker host.

Finally, to indicate the impact and contribution of this research in real scientific use cases, we describe three European projects to which we contribute in various work packages by offering the result of different parts of this research according to their requirements.

8.4 Future Work

The research presented in this thesis focuses on designing and developing a heterogeneous framework to introduce a scalable, fast, and reliable security level for the SDN-based network. To enhance the process of anomaly detection and mitigation process offered by this framework, the following paths can be specified as future work:

- Since using the console as output is inadequate in the production environments, we intend to add a graphical dashboard for the stream processing monitoring to provide an enhanced view of the system processing. For this purpose, we plan to implement Kibana with Elasticsearch as Kibana is an appropriate tool for retrieving the data stored in Elasticsearch via simple queries. Using Kibana, we can monitor network traffic in real time or within a specific time period by displaying the data on a dashboard we have designed.
- Currently, the machine learning approach has a significant role in the IT world, especially in the cybersecurity domain, and several researchers are exploring innovative techniques to improve the performance and accuracy of the machine learning methods; therefore, we consider the Implementation of other machine learning methods, such as deep learning (DL) based on Apache Spark into the proposed framework and providing a performance evaluation with the existing system.

Furthermore, we compare our proposed framework with a traditional firewall system (iptables) in this paper. We also plan to compare our framework with other modern intrusion detection systems (IDS) and intrusion prevention systems (IPS) in order to evaluate its effectiveness.

- The proposed system has been developed on a single computer, not in cluster mode. Spark and Kafka's applications are designed to work in cluster mode. Therefore, we assume that the final result would be more accurate by working in cluster mode in parallel with several machines. One of the primary objectives of this research is to design a framework in which all its components are scalable. Consequently, the computational efficiency of the whole proposed framework will be able to adjust to the increasing load.
- Designing a layer that assists in orchestrating and automating the configuration of all services to make the deployment of the proposed framework easier to manage.

Bibliography

- [1] O. M. E. Committee, "Software-defined networking: The new norm for networks.," ONF White Paper, 2012.
- [2] S.Lou, I.Wu, and B.Pei, "A DEfense MEchanism for Distributed Denial of Service Attack in Software-Defined Networks.," in *Ninth International Conference on Frontier of Computer Science and Technology*, August 2015.
- [3] "Alert(TA14-017A) UDP-based Amplification Attacks," in *US-CERT*, 2014.
- [4] N.N.Dao, J.Park, M.park, and S.Cho, "A feasible method to combat against DDoS attack in SDN network," in *International Conference on Information Networking(ICOIN)*, Cambodia, January 2015.
- [5] H.Polat, O.Polat, A.Cetin, *Detecting DDoS Attacks in Software-Defined Networks Through Feature Selection Methods and Machine Learning Models*, Gazi University: Faculty of Technology, February 2020.
- [6] "Lumen automates DDoS mitigation as attacks surge worldwide," Lumen website, [Online]. Available: https://news.lumen.com/2020-10-28-Lumen-automates-DDoS-mitigation-as-attacks-surge-worldwide#assets_117:20491. [Accessed 20 10 2022].
- [7] wired.com, "Yahoo on trail of site hackers," 8 February 2000. [Online]. Available: <http://www.wired.com>. [Accessed November 2018].
- [8] "Powerful attack cripples internet," 23 October 2002. [Online]. Available: <http://www.greenspun.com>. [Accessed 15 February 2019].
- [9] "Operation Payback cripple Mastercard site in revenge for WikiLeaks ban," 8 December 2010. [Online]. Available: <http://www.gaurdian.co.uk>. [Accessed 15 December 2018].
- [10] P. Criscuolo, "Distributed denial of service:Trin00, Tribe food network, tribe Flood network 2000, and stacheldraht ciac-2319," Technical report, DTIC document, 2000.
- [11] ST Zargar, J Joshi, D Tipper, "A survey of defense mechanisms against distributed dial of service(DDoS) flooding attacks.," in *Comun.Surv*, 2013.
- [12] "Dyn Cyberattack," [Online]. Available: www.theguardian.com/technology/2016/Oct/26/ddos-attack-dyn-mirai-botnet. [Accessed 15 March 2020].
- [13] C Koliass, G Kambourakis, A Stavrou, J Voas, "DDoA in the IoT: Mirai and other botnet," in *Computer*, 2017.
- [14] N.N.Dao, J.Park, M.Park, and S.Cho, "A feasible method to combat against DDoS attack in SDN network," in *International Conference on Information Networking(ICOIN)*, Camodia, January 2015.
- [15] Shu. Z, Wan J, Lin D, Lin J, Vasilakos A.V, imran M, "Security in software-defined networking: Threats and countermeasures.," in *Mob. Netw, Appl*, 2016.
- [16] H.Polat, O.Polat, A.Cetin, "Detecting DDoS Attacks in Software-Defined Networks Through Feature Selection Methods and Machine Learning Models," *Sustainability* 2020.
- [17] A.A.Cardenas, P.K.Manadhata, "Big Data Analytics for Security[J]," in *IEEE Security and Privacy*, Nov-Dec. 2013.

- [18] K. M, "Early Detection and Mitigation of DDoS Attacks In Software Defined Networks.," in *Ryerson University*, Ontario, Canada, 2015.
- [19] K. S. Sahoo, "Detection of Control Layer DDoS Attack using Entropy metrics in SDN : An Empirical Investigation.," in *Ninth International Conference on Advanced Computing (ICoAC)*, 2017.
- [20] Dao, N. N., Park, J., Park, M., & Cho, S., "A feasible method to combat against DDoS attack in SDN network.," in *International Conference on Information Networking*,, January 2015.
- [21] Deepa, V., Sudar, K. M., & Deepalakshmi, P., "Detection of DDoS Attack on SDN Control plane using Hybrid Machine Learning Techniques.," in *International Conference on Smart Systems and Inventive Technology (ICSSIT)*, 2018.
- [22] Tushar Ubale, Ankit Kumar Jain, "Taxonomy of DDoS Attacks in Software-Defined Networking Environment," *FTNCT 2018: Futuristic Trends in Network and Communication Technologies*, no. DOI: 10.1007/978-981-13-3804-5_21, p. 278–291, 25 December 2018.
- [23] Thomas, R. M., James, D., "DDoS Detection and Denial using Third Party Application in SDN.," in *International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, 2017.
- [24] Dharma, N. I. G., Muthohar, M. F., Prayuda, J. D. A., Priagung, K., Choi, D., "Time-based DDoS detection and mitigation for SDN controller," *17th Asia-Pacific Network Operations and Management Symposium: Managing a Very Connected World, APNOMS*, no. <https://doi.org/10.1109/APNOMS.2015.7275389>, p. 550–553, 2015.
- [25] Mousavi, S. M., St-Hilaire, M., "Early Detection of DDoS Attacks Against Software Defined Network Controllers," *Journal of Network and Systems Management*, vol. 26(3), no. <https://doi.org/10.1007/s10922-017-9432-1>, p. 573–591, 2018.
- [26] L. Dhanabal, and S. Shantharajah, "A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms," 2015.
- [27] K Jiang, W Wang, A Wang, and H Wu, "Network Intrusion Detection Combined Hybrid Sampling With Deep Hierarchical Network," February 2020.
- [28] STaghavi Zargar, J Joshi, D Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," in *IEEE Commun. Surveys Tuts*, 2013.
- [29] S. Lim, J. Ha, H. Kim, Y. Kim, and S. Yang, "An SDN-oriented DDoS blocking scheme for botnet-based attacks," in *Sixth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2014.
- [30] R.Kandoi, M. Antikainen, "Denial-of-service attacks in OpenFlow SDN networks.," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015.
- [31] H. Taejin, Y. Seunghyun, A.C. Risdianto, J.W. Kim and H. Lim, "Suspicious Flow Forwarding for Multiple Intrusion Detection Systems on Software-Defined Networks.," in *IEEE Network*, Nov 2016.
- [32] Y. Z. W. L. a. Y. T. H. B. Wang, "DDoS attack protection in the era of cloud computing and Software-defined Networking.," in *Computer Networks*, 2015.

- [33] S. Fichera, L. Galluccio, S. C. Grancagnolo, G. Morabito, and S. Palazzo, "OPERETTA: An OpenFlow-based REmedy to mitigate TCP FLOOD Attacks against web servers," in *Computer Networks*, 2015.
- [34] Peddoju, V. Chouhan and S. K., "Packet monitoring approach to prevent DDoS attack in cloud computing," in *Int. J. Comput. Sci. Electr. Eng. (IJCSEE)*, 2013.
- [35] L Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred, "Statistical approaches to DDoS attack detection and response," in *Proc. DARPA Inf. Survivability Conf. Expo.*, Apr. 2003.
- [36] Nesarani, K. Gurulakshmi and A., "Analysis of IoT Bots against DDoS attack using machine learning algorithm," in *Proc. 2nd Int. Conf. Trends Electron. Inform. (ICOEI)*, May 2018.
- [37] N.-F.Huang, C.-N.Kao, H.-W.Hun, G.-Y.Jai, and C.-L.Lin, "Apply data mining to defense-in-depth network security system," in *Proc. 19th Int. Conf. Adv. Inf. Netw. Appl*, Mar. 2005.
- [38] M. Zekri, S. El Kafhali, N. Aboutabit, and Y. Saadi, "DDoS attack detection using machine learning techniques in cloud computing environments," in *Proc. 3rd Int. Conf. Cloud Comput. Technol. Appl. (CloudTech)*, Oct. 2017.
- [39] J. Wang, D. J. Miller, and G. Kesidis, "Efficient mining of the multidimensional traffic cluster hierarchy for digesting, visualization, and anomaly identification," in *IEEE J. Sel. Areas Commun*, Oct. 2006.
- [40] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R. Mumtaz, and J. González, "Towards sFlow and adaptive polling sampling for deep learning-based DDoS detection in SDN," in *Future Generation Computer Systems*, 2020.
- [41] C. V. G. R. G. Dileep Kumar, "Leveraging Big Data Analytics for Real-time DDoS Attacks Detection in SDN," in *International Journal for Research in Engineering Application and Management (IJREAM)*, May 2018.
- [42] D. C.-T. L. K. Q. T Zhao, "A Neural-Network Based DDoS Detection System Using Hadoop and HBase," in *IEEE International Conference on High-Performance Computing and Communications (HPCC)*, 2015.
- [43] S. Hameed, U. Ali, "HADEC: Hadoop-based live DDoS detection framework," *EURASIP Journal on Information Security*, 2018:11.
- [44] A. M. Karimi, Q. Niyaz, W. Sun, A. Y. Javaid, and V. K. Devabhaktuni, "Distributed network traffic feature extraction for a real-time ids," in *Electro Information Technology (EIT)*, 2016.
- [45] A. Gupta, R. Birkner, M. Canini, N. Feamster, C. Mac-Stoker, and W. Willinger, "Network monitoring as a streaming analytics problem," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 2016.
- [46] Y Lee, W Kang, Y Lee, "Traffic Monitoring and Analysis of Lecture Notes in Computer Science, ed. by J Domingo-Pascual, Y Shavitt, and S Uhlig. A hadoop-based packet trace processing tool," Berlin Heidelberg, 2011.
- [47] Y Lee, Y Lee, "Detecting DDoS attacks with Hadoop," in *Proceedings of The ACM CoNEXT Student Workshop (CoNEXT '11 Student)*, New York, 2011.
- [48] H. H. a. A. A. H. Benbrahim, "Comparison between Hadoop and Spark," in *the International Conference on Industrial Engineering and Operations Management*, Bangkok, Thailand, March 5-7, 2019.

- [49] A. Saied, R. E. Overill, and T. Radzik, "Detection of known and un-known DDoS attacks using artificial neural networks," *Neurocomputing*, vol. 172, p. 385–393, Jan. 2016.
- [50] P. Dahiya, D. Srivastava, "Network Intrusion Detection in Big Dataset Using Spark," *Procedia Computer Science* 132, pp. 253-262.
- [51] M. Belouch, S. El Hadaj, and M. Idhammad, "Performance evaluation of intrusion detection based on machine learning using Apache Spark.," in *Procedia Computer Science* 127, 2018.
- [52] F Karataş, S A.Korkmaz, "BigData: Controlling Fraud by Using Machine Learning Libraries on Spark," *International Journal of Applied Mathematics, Electronics, and Computers Advanced Technology and Science*, no. ISSN:2147-822821, DOI: 10.18100/ijamec.2018138629.
- [53] N V Patil, C.R Krishna, and K Kumar, "S-DDoS: Apache spark based real-time DDoS detection system," *Journal of Intelligent Fuzzy Systems*, no. DOI:103233/JIFS-179733, IOS Press, ISSN 1064-1246/20/\$35.00, 2020.
- [54] M A Manzoor, Y Morgan, "Real-time support vector machine-based network intrusion detection system using Apache Storm," in *IEEE 7th annual information technology, electronics and mobile communication conference (IEMCON)*, 2016.
- [55] K Peng, V C. M. Leung, and L Zheng, "Intrusion Detection System Based on Decision Tree over Big Data in Fog Environment," *Hindawi Wireless Communication and Mobile Computing*, no. Article ID 4680867, <https://doi.org/10.1155/2018/4680867>, 2018.
- [56] Zhao J, Chen S, Cao M, Cui B, "Malware algorithm classification method based on big data analysis," in *International Journal of Web and Grid Services.*, 2017.
- [57] Terzi, D., Terzi, R. and Sagiroglu, S, "Big data analytics for network anomaly detection from netflow data," in *International Conference on Computer Science and Engineering (UBMK)*, Antalya, 2017.
- [58] A. Alsirhani, S. Sampalli, and P. Bodorik, "DDoS attack detection system: Utilizing classification algorithms with Apache spark," in *Proc. 9th IFIP Int. Conf. New Technol. Mobility Secur. (NTMS)*, Feb. 2018.
- [59] Ying Gao, Honngrui Wu, Binjie Song, Y Jin, X Luo, and Xing Zeng, "A Distributed Network Intrusion Detection System for Distributed Denial of Service Attacks in Vehicular Ad Hoc Network," 2019.
- [60] Leon-Garcia, A. Tizghadam and A., "Application platform for smart transportation, pp. .," in *Future Access Enablers of Ubiquitous and Intelligent Infrastructures.*, 2015.
- [61] S. Arora, "Analyzing mobile phone usage using clustering in spark mllib and pig," 2017.
- [62] Sundararajan, R., "Software-Defined Networking (SDN)," 2013.
- [63] Y. Li, D. Li, W. Cui, and R. Zhang, "Research-based on OSI model," in *2011 IEEE 3rd International Conference on Communication Software and Networks*, May 2011.
- [64] T. Socolofsky, C. Kale, "A TCP/IP Tutorial," January 1991. [Online]. Available: <https://tools.ietf.org/html/rfc1180>. [Accessed 14 5 2022].
- [65] T. Benson, A. Akella, and D. Maltz, "Unraveling the complexity of network management," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, 2009.

- [66] D. Kreutz, F. M. V. Ramos, P. E. VerAˆssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," in *Proceedings of the IEEE*, Jan 2015.
- [67] C. Martinez, R. Ferro, and W. Ruiz, "Next-generation networks under the SDN and open-flow protocol architecture," in *2015 Workshop on Engineering Applications - International Congress on Engineering (WEA)*, Oct 2015.
- [68] S. Wang, D. Li, and S. Xia, "The problems and solutions of network update in SDN: A survey," in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2015.
- [69] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: From concept to implementation," in *IEEE Communications Surveys Tutorials*, 2014.
- [70] K.Gray, T.D.Nadeau, "SDN: Software Defined Networks, Chapter 6, Data Center Concepts and Constructs," vol. Chapter 6, O'REILLY, 2013.
- [71] D. Kreutz, F. Ramos, "Software-Defined Networking: A Comprehensive Survey," in *Proceedings of the IEEE*, 2015.
- [72] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," in *IEEE Communications Surveys & Tutorials*, 2014.
- [73] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," in *ACM SIGCOMM Computer Communication Review*, 2008.
- [74] M. Shin, K. Nam, and H. Kim, "Software-defined networking (SDN): A reference architecture and open APIs," in *2012 International Conference on ICT Convergence (ICTC)*, Oct 2012.
- [75] L. Richardson, S. Ruby, RESTful Web Services. Web Services for the Real World., Mayo: O'Reilly Media, 2007.
- [76] Foster, N., "Frenetic: A network programming language," in *Proceedings of the 16th ACM SIGPLAN international conference on Functional programming*, September 2011.
- [77] H. Kim, N. Feamster., "Improving network management with software defined networking," in *IEEE Communications Magazine*, 2013.
- [78] T. Hinrichs, N. Gude, M. Casado, "Practical declarative network management," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, August 2009..
- [79] C. Monsanto, N. Foster, R. Harrison, "A compiler and run-time system for network programming languages," in *Proceedings of the 39th annual ACM SIGPLAN-SIGACT Symposium on Principles of programming languages*, 2012.
- [80] J. Reich, C. Monsanto, N. Foster, "Modular SDN programming with pyretic," in *Usenix, The Advanced Computing Systems Association*, Octubre, 2013.
- [81] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Secure your northbound SDN API," in *2015 Seventh International Conference on Ubiquitous and Future Networks*, July 2015.
- [82] M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, attributes, and use cases : A compass for SDN," in *IEEE Communications Magazine*, June 2014.
- [83] H. Yin, "SDNi: A message exchange protocol for software-defined networks (SDNS) across multiple domains.," in *Internet Engineering Task Force, Internet Draft.*, 2012.

- [84] D. Kreutz, F. Ramos, "Software-Defined Networking: A Comprehensive Survey," in *Proceedings of the IEEE*, 2015.
- [85] M. Lessing, "What Are SDN Southbound APIs?," SDxCentral Studios, September 2019. [Online]. Available: <https://www.sdxcentral.com/resources/sdn/southbound-interface-api>. [Accessed 10 6 2022].
- [86] 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," in *IEEE Communications Magazine*, July 2013.
- [87] J. Case, M. Fedor, "A Simple Network Management Protocol (SNMP). .," 2022 May 1990.. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc1098>. [Accessed 10 6].
- [88] R. Enns, M. Bjorklund, J. Schoenwaelder, "Network Configuration Protocol (NETCONF)," RFC 6241. IETF, June 2011.. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc6241>. [Accessed 10 6 2022].
- [89] Y. Rekhter, T. Lid, S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, January 2006. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc4271>.
- [90] B. Pfaff, B. Davie, "The Open vSwitch Database Management Protocol.," RFC 7047. IETF, December 2013.. [Online]. Available: <https://www.ietf.org/rfc/rfc7047.txt>.
- [91] M. Smith, M. Dvorkin, Y. Laribi, V. Pandey, "OpFlex Control Protocol," IETF, 2 April 2014. [Online]. Available: <http://tools.ietf.org/pdf/draft-smith-opflex-00.pdf>. [Accessed 10 6 2022].
- [92] A. Doria, J. Hadi, R. Hass, H. Khosravi, "Forwarding and Control Element Separation (ForCES) Protocol Specification.," RFC 5810, March 2010.. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc5810>. [Accessed 10 6 2022].
- [93] N. McKeown, "OpenFlow: Enabling Innovation in Campus Networks," Stanford University, March, 2008.
- [94] M. K. Jaiswal, "Innovations in Software-Defined Networking and Network Functions Virtualization," no. 10.4018/978-1-5225-3640-6.ch003, p. 20, 2018.
- [95] H. Egilmez, S. Dane, "OpenQoS: OpenFlow controller design and test network for multimedia delivery with quality of service," in *Proc. NEM Summit, Implementing Future Media Internet Towards New Horizons*, December 2012.
- [96] C-Hao, Chang and Y-Dar Lin, "OpenFlow Version Roadmap," 11 September 2015. [Online]. Available: http://speed.cis.nctu.edu.tw/~ydlin/miscpub/indep_frank.pdf. [Accessed 6 2022].
- [97] "OpenFlow Switch Specification, Version 1.0.0 Implemented (Wire Protocol 0x01)," December, 2009.
- [98] D. Kreutz, F. Ramos, "Software-Defined Networking: A Comprehensive Survey," in *Proceedings of the IEEE*, January, 2015.
- [99] "Openflow Spec 1.3 Coverage," 22 Feb. 2017. [Online]. Available: <https://seagullbird.xyz/posts/openflow-1.3-coverage/>. [Accessed 10 6 2022].
- [100] A. C. Jimenez, "LOGICAL SECURITY. Information management: Confidentiality, integrity, availability and tightness," 31 March 2017. [Online]. Available: <https://cronicaseguridad.com/2017/03/31/seguridad-logica-gestion-la-informacion-confidencialidad-integridad-disponibilidad-estaqueidad/>. [Accessed 15 6 2022].

- [101] Zhen Yan, Peng Zhang, Athanasios V. Vasilakos, "A security trust framework for virtualized networks and software-defined networking", 2015., " *Wiley Online Library*, vol. 9, no. <https://doi.org/10.1002/sec.1243>, pp. 3059-3069, 26 March 2015.
- [102] S.-H. S. Sezer, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller y N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," in *Communications Magazine*, 2013.
- [103] Ganjali, A. Tootoonchian and Y., "HyperFlow: A Distributed Control Plane for Open-Flow," in *Proc. 2010 Internet Network Management Conf. Research on Enterprise Networking*, 2010.
- [104] Andrés Felipe Murillo, Sandra Rueda, Laura Victoria Morales, Álvaro Cardenas, "SDN and NFV Security: Challenges for Integrated Solutions," no. DOI: 10.1007/978-3-319-64653-4_3, 12 November 2017.
- [105] Juan Felipe Botero, Juan Camilo Correa Chica, Jenny Cuatindioy Imbachi, "Security in SDN: A comprehensive survey," *Vega Journal of Network and Computer Applications*, 2018.
- [106] Ahmad, I., Namal, S., Ylianttila, M., & Gurtov, A., "Security in Software Defined Networks: A Survey," in *IEEE Communications Surveys & Tutorials*, 2015.
- [107] A. Linguori, M. Winandy, "The Diamond Approach for SDN Security," in *IEEE Softwarization*, March, 2018.
- [108] D. Kreutz, F. Ramos, P. Verissimo, "Towards Secure and Dependable Software-Defined Networks," in *ACM*, August, 2013.
- [109] Z. Yan, R. MaLavery, "Autonomic Trust Management in a Component-Based Software System," in *Proceedings of the 3rd International Conference on Autonomic and Trusted Computing (ATC2006)*, 2006.
- [110] Y. Desmedt, Y. Franke, "Threshold Cryptosystems," in *Proceedings of the 9th Annual International Cryptology Conference.*, August, 1989.
- [111] Schehlmann, L., Abt, S., Baier, H., "Blessing or curse? revisiting security aspects of software-defined networking. In:," in *10th International Conference on Network and Service Management (CNSM) and Workshop*, 2014.
- [112] "2016 Cost of Cyber Crime Study & the Risk of Business Innovation," Ponemon Institute, October 2016. [Online]. Available: <https://www.ponemon.org/local/upload/file/2016%20HPE%20CCC%20GLOBAL%20REPORT%20FINAL%203.pdf>. [Accessed 10 6 2022].
- [113] P. J. Criscuolo, "Distributed Denial of Service,," in *Tribe Flood Network 2000, and Stacheldraht CIAC-2319*, February 14, 2000.
- [114] "What is a DDoS attack?," Cloudflare, [Online]. Available: <https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/>.
- [115] D Warburton, E Ojeda, "DDoS Attack Trends for 2020," 07 May 2021. [Online]. Available: <https://www.f5.com/labs/articles/threat-intelligence/ddos-attack-trends-for-2020>. [Accessed 15 4 2022].
- [116] Sobana Sikkanan, Kasthuri M., "Denial-of-Service and Botnet Analysis, Detection, and Mitigation," 2021 , pp. 114-151.
- [117] J. Paracuellos, "Proactive and reactive defense against DDoS attacks in a simulated environment of software-defined networks," 2016.

- [118] Radware, "Attack Types and Their Effects," in *DDoS Survival Handbook*, 2013.
- [119] Dong, S., Abbas, K., & Jain, R., "A Survey on Distributed Denial of Service (DDoS) Attacks in SDN and Cloud Computing Environments.," 2019.
- [120] Lawal, B. H., & At, N., " Real-Time Detection and Mitigation of Distributed Denial of Service (DDoS) Attacks in Software-Defined Networking (SDN)," in *26th Signal Processing and Communications Applications Conference (SIU)*, 2018.
- [121] Mousavi, S. M., & St-Hilaire, M., "Early Detection of DDoS Attacks Against Software Defined Network Controllers," in *Journal of Network and Systems Management*, 2018.
- [122] Haque, M. R., Ali, S., Tan, S. C., Yusoff, Z., Kwang, L. C., Kaspin, I. R., & Ziri, S. R., "The Motivation of DDoS Attack-Aware in Software-Defined Networking Controller Placement," in *International Conference on Computer and Applications, ICCA 201*, 2017.
- [123] N. Z. & S. J. A. Bawany, "Application Layer DDoS Attack Defense Framework for Smart City using SDN," May 2016.
- [124] Ahmad, I., Namal, S., Ylianttila, M., & Gurtov, A., "Security in Software Defined Networks: A Survey," 2015.
- [125] THENEWSTACK, "SDN Series Part Three: NOX, the Original OpenFlow Controller," 15 December 2014. [Online]. Available: <https://thenewstack.io/sdn-series-part-iii-nox-the-original-openflow-controller/>. [Accessed 15 3 2022].
- [126] "SDN Series Part Four: Ryu, a Rich-Featured Open Source SDN Controller Supported by NTT Labs," THENEWSTACK, 23 December 2014. [Online]. Available: <https://thenewstack.io/sdn-series-part-iv-ryu-a-rich-featured-open-source-sdn-controller-supported-by-ntt-labs/>. [Accessed 15 5 2022].
- [127] Asadollahi, Saleh & Goswami, Bhargavi, "Experimenting with the scalability of floodlight controller in software-defined networks," no. 10.1109/ICEECCOT.2017.8284684., pp. 288-292, 2017.
- [128] "REST API Tutorial," 7 April 2022. [Online]. Available: <https://restfulapi.net/> . [Accessed 20 June 2022].
- [129] Z. K. Khattak, M. Awais and A. Iqbal, "Performance evaluation of OpenDaylight SDN controller," in *20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2014.
- [130] "What's Software Defined Networking (SDN)? Definition," SDxCentral Studios, 26 August 2016. [Online]. Available: <https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/>. [Accessed 15 April 2022].
- [131] "ONOS - a new carrier-grade SDN network operating system," Open Networking Foundation (ONF), [Online]. Available: <https://onosproject.org/>. [Accessed 20 May 2022].
- [132] A. Al-Shabibi, "ONOS Project. "ONOS Platform Architecture"," OpenDayLight, 4 March 2015. [Online]. Available: <https://www.slideshare.net/OpenDaylight/onos-platform-architecture>. [Accessed 10 June 2022].
- [133] "SDN Series Part Seven: ONOS," THENEWSTACK, 3 March 2015. [Online]. Available: <https://thenewstack.io/open-source-sdn-controllers-part-vii-onos/>. [Accessed 10 May 2022].
- [134] O. Salman, I. Elhajj, A. Chehab, "SDN controllers: A comparative study," in *18th Mediterranean Electrotechnical Conference (MELECON)*, 2016.

- [135] "ONOS Project," [Online]. Available: <https://wiki.onosproject.org/>. [Accessed 20 6 2022].
- [136] A. Koshibe, 30 March 2016. [Online]. Available: <https://wiki.onosproject.org/display/ONOS15/Use+Cases>. [Accessed 10 May 2022].
- [137] S. Secci, S. Scott-Hayward, Q. Pham Van, D. Verchere, A. Sow, C. Basquin, D. Smyth, K. Attou, "ONOS Security & Performance Analysis (Report No. 2)," Open Networking Foundation,, November 2018.
- [138] A. Koshibe, "The ONOS CLI," [Online]. Available: <https://wiki.onosproject.org/display/ONOS/The+ONOS+CLI>. [Accessed 20 6 2022].
- [139] A. Koshibe, "The ONOS Web GUI," [Online]. Available: <https://wiki.onosproject.org/display/ONOS/The+ONOS+Web+GUI>. [Accessed 20 6 2022].
- [140] "OSGi Framework Overview," [Online]. Available: <http://docs.osgi.org/specification/osgi.core/7.0.0/ch01.html>. [Accessed 20 6 2022].
- [141] P. Paralogarajah, "OSGi in a Nutshell," 9 Sept 2017. [Online]. Available: <https://piraveenaparalogarajah.medium.com/osgi-in-a-nutshell-aafc3a86cff0>. [Accessed 30 4 2022].
- [142] E. Olkhovskaya, "ONOS Cluster Coordination," [Online]. Available: <https://wiki.onosproject.org/display/ONOS/Cluster+Coordination>. [Accessed 20 6 2022].
- [143] A. Koshibe, "ONOS System Components," [Online]. Available: <https://wiki.onosproject.org/display/ONOS/System+Components>. [Accessed 20 6 2022].
- [144] T. Vachuska, "Overview of ONOS architecture," [Online]. Available: <https://wiki.onosproject.org/display/ONOS/Overview+of+ONOS+architecture>. [Accessed 20 6 2022].
- [145] Peterson, Cascone, O'Connor, Vachuska, and Davie, "Software-Defined Networks: A Systems Approach (Chapter 6: Network OS)," [Online]. Available: <https://sdn.systemsapproach.org/onos.html>. [Accessed 20 6 2022].
- [146] Arvind, "Maven Tutorial: All You Need To Know To Get Started.," [Online]. Available: <https://www.edureka.co/blog/maven-tutorial/>. [Accessed 20 12 2022].
- [147] E. Z. Moghaddam, "Getting to know Apache Karaf," [Online]. Available: <http://developmentor.blogspot.com/2013/12/getting-to-know-apache-karaf.html>. [Accessed 20 6 2022].
- [148] S. Rao, "SDN Series Part Seven: ONOS.," [Online]. Available: <https://thenewstack.io/open-source-sdn-controllers-part-vii-onos/>. [Accessed 20 12 22].
- [149] M. Virk, "Classification Metrics," 15 Feb. 2020. [Online]. Available: <https://medium.com/@m.virk1/classification-metrics-65b79bfdd776>. [Accessed 15 6 2022].
- [150] 26 Aug. 2016. [Online]. Available: <http://kflu.github.io/2016/08/26/2016-08-26-visualizing-precision-recall/>. [Accessed 15 6 2022].
- [151] M. Mohri, A. Rostamizadeh, and A. Talwalkar, Foundations of Machine Learning, 2nd edition, London, England: The MIT Press, 2018.

- [152] S. Kiourkoulis, "DDoS datasets, Use of machine learning to analyse intrusion detection performance," Department of Computer Science, Electrical and Space Engineering,, 2020. [Online].
- [153] Preeti Mishra, Vijay Varadharajan, Uday Tupakula, Emmanuel S. Pilli., "A Detailed Investigation and Analysis of using Machine Learning Techniques for Intrusion Detection," in *IEEE Communications Surveys & Tutorials*.
- [154] T. M. Mitchell, Machine Learning, McGraw-Hill Science/Engineering/Math, 1997.
- [155] S. M. Kasongo, "Development and Evaluation of a Deep Learning Based Intrusion Detection Model for Wireless Networks," in *Doctoral Thesis*, 2020.
- [156] D. Denning, "An Intrusion Detection Model," in *IEEE Transactions on Software Engineering*, 1987.
- [157] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," in *IEEE Security and Privacy Symposium*, May 2010.
- [158] "KDD Cup 1999 Data," in *University of California, Irvine*, 1999.
- [159] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015.
- [160] M. Tavallaei, E. Bagheri, W. Lu, A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Data Set," in *Submitted to Second IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, 2009.
- [161] J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," in *ACM Transactions on Information and System Security*, 2000.
- [162] F. Salo, M. Injadat, A. B. Nassif, A. Shami, A. Essex, "Data Mining Techniques in Intrusion Detection Systems: A Systematic Literature Review," in *IEEE Communications Surveys & Tutorials*, 2018.
- [163] L. Dhanabal, S.P. Shantharajah, "A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms," in *International Journal of Advanced Research in Computer and Communication Engineering*, June 2015.
- [164] N. K. S. Choudhary, "Analysis of KDD-Cup'99, NSL-KDD and UNSW-NB15 Datasets using Deep Learning in IoT," in *International Conference on Computational Intelligence and Data Science (ICCIDS 2019)*, 2019.
- [165] T. Tang, L. Mhamdi, D. McLernon, Z. S.A.R. and M. M. Ghogho, "Deep Learning Approach for Network Intrusion Detection in Software Defined Networking," in *Proc. International Conf. on Wireless Networks and Mobile Communications*, Morocco, 26-29 Oct.2016.
- [166] Latah, Majd & Toker, Levent, "An Efficient Flow-based Multi-level Hybrid Intrusion Detection System for Software-Defined Networks," 2018.
- [167] N. Moustafa, "The UNSW-NB15 Dataset," Intelligent Security Group(ISG), UNSW, Australia, [Online]. Available: <https://research.unsw.edu.au/projects/unsw-nb15-dataset>. [Accessed 20 6 2022].
- [168] Zeinab Zoghi and Gursel Serpen, "UNSW-NB15 Computer Security Dataset: Analysis through Visualization," in *Electrical Engineering & Computer Science*, Ohio, USA.
- [169] "Mininet overview," [Online]. Available: <http://mininet.org/overview>. [Accessed 20 6 2022].

- [170] S. Studios, "What Is Open vSwitch (OVS)?," [Online]. Available: <https://www.sdxcentral.com/cloud/open-source/definitions/what-is-open-vswitch/>. [Accessed 25 6 2022].
- [171] L. Foundation. [Online]. Available: <https://www.openvswitch.org/>. [Accessed 25 6 2022].
- [172] "Open vSwitch," [Online]. Available: https://en.wikipedia.org/wiki/Open_vSwitch. [Accessed 25 6 2022].
- [173] "hping3," 14 Sept. 2021. [Online]. Available: <https://www.kali.org/tools/hping3/#:~:text=hping3%20is%20a%20network%20tool,transfer%20files%20under%20supported%20protocols..> [Accessed 20 6 2022].
- [174] "htop - an interactive process viewer," [Online]. Available: <https://htop.dev/>. [Accessed 20 6 2022].
- [175] "Apache Spark, a unified analytics engine for large-scale data processing," [Online]. Available: <https://spark.apache.org/>. [Accessed 20 6 2022].
- [176] Shanjiang Tang, Bingsheng He, Ce Yu, Yusen Li, Kun Li,, "A Survey on Spark Ecosystem: Big Data Processing Infrastructure, Machine Learning, and Applications," in *IEEE Transactions On Knowledge and Data Engineering*, Feb. 2020.
- [177] S. Banerjee, "Introduction to Apache Spark," 7 2018. [Online]. Available: <https://www.kdnuggets.com/2018/07/introduction-apache-spark.html>. [Accessed 20 4 2022].
- [178] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., "Mllib: Machine learning in apache spark," (2015).
- [179] R. D. X. C. P. X. P. J. Z. H. S. Salloum, "Big data analytics on Apache Spark," in *Int J Data Sci Anal*, 2016.
- [180] "Spark Streaming Tutorial for Beginners," [Online]. Available: <https://data-flair.training/blogs/apache-spark-streaming-tutorial/>. [Accessed 20 6 2022].
- [181] A. P. Gulati, "Introduction to Spark Streaming," 14 August 2022. [Online]. Available: <https://www.analyticsvidhya.com/blog/2022/08/introduction-to-spark-streaming/>. [Accessed 15 11 2022].
- [182] "Apache Kafka," [Online]. Available: <https://kafka.apache.org/intro>. [Accessed 20 6 2022].
- [183] L. Johansson, "Apache Kafka for beginners - What is Apache Kafka? " 19 3 2020. [Online]. Available: <https://www.cloudkarafka.com/blog/part1-kafka-for-beginners-what-is-apache-kafka.html>. [Accessed 20 6 2022].
- [184] "Kafka Architecture and Its Fundamental Concepts," [Online]. Available: <https://data-flair.training/blogs/kafka-architecture/>. [Accessed 15 11 2022].
- [185] "Kafka Architecture," [Online]. Available: <http://clouduable.com/blog/kafka-architecture/index.html>. [Accessed 15 11 2022].
- [186] "Apache ZooKeeper," [Online]. Available: <https://zookeeper.apache.org/>. [Accessed 20 6 2022].
- [187] "What is Elasticsearch?," [Online]. Available: <https://www.elastic.co/what-is/elasticsearch>. [Accessed 20 6 2022].

- [188] "SparkContext," [Online] Available: <https://spark.apache.org/docs/3.1.1/api/python/reference/api/pyspark.SparkContext.html>. [Accessed 20 6 2022].
- [189] "Spark Streaming Programming Guide," [Online]. Available: <https://spark.apache.org/docs/latest/streaming-programming-guide.html>. [Accessed 20 6 2022].
- [190] "RDD Programming Guide," [Online]. Available: <https://spark.apache.org/docs/latest/rdd-programming-guide.html>. [Accessed 24 6 2022].
- [191] D. Rai, "Feature Engineering in pyspark," [Online]. Available: <https://dhiraj-rai.medium.com/essentials-of-feature-engineering-in-pyspark-part-i-76a57680a85>. [Accessed 24 6 2022].
- [192] "ML - Features," [Online]. Available: <https://spark.apache.org/docs/1.4.1/ml-features.html>. [Accessed 24 6 2022].
- [193] H. Hajjalian, C. TOMA, "Network Anomaly Detection by Means of Machine Learning: Random Forest Approach with Apache Spark", in *Informatics Economics*, 2018.
- [194] "Mininet Python API Reference Manual," [Online]. Available: <http://mininet.org/api/index.html>. [Accessed 25 6 2022].
- [195] "NEPHELE project," 2018. [Online]. Available: <http://www.nepheleproject.eu/>. [Accessed 5 3 2022].
- [196] "SEcure Networking for a DATa center cloud in Europe," 2016. [Online]. Available: <https://www.celticnext.eu/project-sendate/>. [Accessed 5 3 2022].
- [197] "ONAP," [Online]. Available: <https://www.onap.org>.
- [198] "Project AI-NET-PROTECT," 2021. [Online]. Available: <https://www.celticnext.eu/project-ai-net-protect/>. [Accessed 5 5 2022].
- [199] A. P. T. K. Y. H. K. W. a. M. M. H. Z. Ye, "Sparktext: Biomedical text mining on big data framework," in *PloS one*.
- [200] M. A. Uddin, J. bibi Joolee, A. Alam, and Y.-K. Lee, "Human action recognition using adaptive local motion descriptor in spark," 2017.
- [201] A. Campanella, "ONOS Southbound: Protocol, Providers, Drivers," [Online]. Available: <https://wiki.onosproject.org/display/ONOS/Southbound%3A+Protocol%2C+Providers%2C+Drivers>. [Accessed 20 6 2022].
- [202] A. P. Tafti, E. Behraves, M. Assefi, E. LaRose, J. Badger, J. Mayer, A. Doan, D. Page, and P. Peissig, "bignn: an open-source big data toolkit focused on biomedical sentence classification," in *Proceedings of the IEEE BIG DATA*, 2017.
- [203] E. R. Sparks, A. Talwalkar, D. Haas, M. J. Franklin, M. I. Jordan, and T. Kraska, "Automating model search for large scale machine learning," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, 2015.
- [204] A. Hryhorzhevskaya, M. Wiewiórka, M. Okoniewski, and T. Gambin, "Scalable framework for the analysis of population structure using the next generation sequencing data," in *International Symposium on Methodologies for Intelligent Systems.*, 2017.
- [205] M.-S. Lee, E. Kim, C.-S. Nam, and D.-R. Shin, "Design of educational big data application using spark," in *Advanced Communication Technology (ICACT)*, 2017.

- [206] A.Saied, R.E.Overill, and T.Radzik, "Detection of known and unknown DDoS attacks using artificial neural networks," in *Neurocomputing*, Jan. 2016.
- [207] T. Zhao, D. C.-T. Lo, and K. Qian, "A neural-network based DDoS detection system using Hadoop and HBase," in *Proc. IEEE Int. Conf. High Perform. Comput. Commun*, Aug. 2015.
- [208] T Zhao, D Chia-Tien Lo, K Qian, "A Neural-Network Based DDoS Detection System Using Hadoop and HBase," in *IEEE International Conference on High-Performance Computing and Communications (HPCC)*, 2015.
- [209] G. Dileep Kumar, C V Guru Rao, "Leveraging Big Data Analytics for Real-time DDoS Attacks Detection in SDN," in *International Journal for Research in Engineering Application and Management (IJREAM)*, May 2018.

Appendix A

A. The Source Code of Machine Learning Evaluation

```
1 #import libraries
2
3 import os
4 import numpy as np
5 import pandas as pd
6 from sklearn.model_selection import train_test_split
7 from sklearn.linear_model import LogisticRegression
8 from sklearn.tree import DecisionTreeClassifier
9 from sklearn.ensemble import RandomForestClassifier
10 from sklearn.naive_bayes import GaussianNB
11 from sklearn.neighbors import KNeighborsClassifier
12 from sklearn import metrics
13 from sklearn.metrics import accuracy_score
14 from sklearn.metrics import precision_score
15 from sklearn.metrics import recall_score
16 from sklearn.metrics import confusion_matrix
17 import matplotlib.pyplot as plt; plt.rcdefaults()
18 import matplotlib.pyplot as plt
19 # Applying k-Fold Cross Validation
20 from sklearn.model_selection import cross_val_score
21 from IPython.display import display
22
23 os.system("clear")
24
25 #importing dataset
26
27 datafile_handler = open("/home/gwdguser/Downloads/source/ML-NSLKDD/KDDTrain+.csv", "r")
28
29 # creating a Pandas DataFrame using read_csv function
30 # load data from a csv file.
31 dataset = pd.read_csv(datafile_handler, sep = ",")
32
33 # closing the datafile handler
34 datafile_handler.close()
35
36 # creating a dict file
37 protocol_type = {'tcp': 1, 'udp': 2, 'icmp': 3}
38 flag = { 'OTH':1, 'REJ':2, 'RST0':3, 'RST0S0':4, 'RSTR':5, 'S0':6, 'S1':7, 'S2':8, 'S3':9, 'SF':10, 'SH':11}
39 service = [{"aol":1, 'auth':2, 'bgp':3, 'courier':4, 'csnet_ns':5, 'ctf':6, 'daytime':7, 'discard':8, 'domain':9,
40 'domain_u':10, 'echo':11, 'eco_i':12, 'ecr_i':13, 'efs':14, 'exec':15, 'finger':16, 'ftp':17, 'ftp_data':18,
41 'gopher':19, 'harvest':20, 'hostnames':21, 'http':22, 'http_2784':23, 'http_443':24, 'http_8001':25, 'imap4':26,
42 'IRC':27, 'iso_tsap':28, 'klogin':29, 'kshell':30, 'ldap':31, 'link':32, 'login':33, 'mtp':34, 'name':35,
43 'netbios_dgm':36, 'netbios_ns':37, 'netbios_ssn':38, 'netstat':39, 'nntp':40, 'nntp':41, 'ntp_u':42, 'other':43,
44 'pm_dump':44, 'pop_2':45, 'pop_3':46, 'printer':47, 'private':48, 'red_i':49, 'remote_job':50, 'rje':51, 'shell':52,
45 'smtp':53, 'sql_net':54, 'ssh':55, 'sunrpc':56, 'supdup':57, 'systat':58, 'telnet':59, 'tftp_u':60, 'tim_i':61,
46 'time':62, 'urh_i':63, 'urp_i':64, 'uucp':65, 'uucp_path':66, 'vmnet':67, 'whois':68, 'X11':69, 'Z39_50':70}]
47
48 # traversing through dataframe
49 # protocol_type, flag, service column and writing
50 # values where key matches
51 dataset.protocol_type = [protocol_type[item] for item in dataset.protocol_type]
52 dataset.flag = [flag[item] for item in dataset.flag]
53 dataset.service = [service[item] for item in dataset.service]
54 print(dataset)
55
56 #printing the head of the dataset
57 dataset.head()
58
59
60 #splitting dataset into features and class
61 X = dataset.iloc[:, 0:41].values
62 y = dataset.iloc[:, 41].values
63
64
65 #splitting dataset into train and test set
66 from sklearn.model_selection import train_test_split
67 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
68
69
```



```

70 # Training the Selected Algorithms
71 # Create the RandomForest Model
72 RFclassifier = RandomForestClassifier(n_estimators=200, bootstrap = True, max_features = 'sqrt')
73 # Create the Decision Tree Model
74 DTclassifier = DecisionTreeClassifier(criterion = "entropy", max_depth=10)
75 # Create the Naive Bayes Model
76 NBclassifier = GaussianNB(priors=None, var_smoothing=1e-09)
77 # Create the Logistic Regression Model
78 LRclassifier = LogisticRegression(solver = 'lbfgs', max_iter=100)
79 # Create the KNeighbors Model
80 KNNclassifier = KNeighborsClassifier(n_neighbors=5)
81
82 # Train the models using the training sets
83 # Training Random Forest Algorithm
84 RFclf = RFclassifier.fit(X_train, y_train)
85 # Training Decision Tree Algorithm
86 DTclf = DTclassifier.fit(X_train, y_train)
87 # Training Naive Bayes Algorithm
88 NBclf = NBclassifier.fit(X_train, y_train)
89 # Training Logistic Regression Algorithm
90 LRclf = LRclassifier.fit(X_train, y_train)
91 # Training KNeighbors Algorithm
92 KNNclf = KNNclassifier.fit(X_train, y_train)
93
94
95 #predict the response for test dataset
96 #for RandomForest Algorithm
97 RFPred = RFclf.predict(X_test)
98 #for decision tree Algorithm
99 DTPred = DTclf.predict(X_test)
100 #for Naive Bayes Algorithm
101 NBPred = NBclf.predict(X_test)
102 #for Logistic Regression Algorithm
103 LRPred = LRclf.predict(X_test)
104 #for KNeighbors Algorithm
105 KNNPred = KNNclassifier.predict(X_test)
106
107 #print("Accuracy:",metrics.accuracy_score(y_test, RFPred))
108 #print("Precision:",metrics.precision_score(y_test, RFPred))
109 #print("Recall:",metrics.recall_score(y_test, RFPred))
110
111 #Calculate the Confusion Matrix.
112 #for RandomForest
113 RFcm = confusion_matrix(y_test,RFPred)
114 print("RF Confusion Martix")
115 print(RFcm)
116 print("-----")
117 #for decision tree
118 DTcm = confusion_matrix(y_test,DTPred)
119 print("DT Confusion Martix")
120 print(DTcm)
121 print("-----")
122 #for Naive Bayes
123 NBcm = confusion_matrix(y_test,NBPred)
124 print("NB Confusion Martix")
125 print(NBcm)
126 print("-----")
127 #for Logistic Regression
128 LRCm = confusion_matrix(y_test,LRPred)
129 print("LR Confusion Martix")
130 print(LRCm)
131 print("-----")
132 #for KNeighbors Algorithm
133 KNNcm = confusion_matrix(y_test,KNNPred)
134 print("KNN Confusion Martix")
135 print(KNNcm)
136 print("-----")
137

```

```

138 #Calculate the Accuracy, Recall, and Precision
139 #for RandomForest
140 RFacc = 100*accuracy_score(y_test,RFpred)
141 print("-----")
142 print("RF accuracy is: ", RFacc)
143
144 RFrecall=RFcm[1,1]/(RFcm[1,0]+RFcm[1,1])
145 print("RF Recall is: "+ str(RFrecall))
146 RFprecision= RFcm[1,1]/(RFcm[1,0]+RFcm[0,1])
147 print("RF Precision is: "+ str(RFprecision))
148 print("RF F-measure is: "+ str(2*((RFprecision*RFrecall)/(RFprecision+RFrecall))))
149
150 #for decision tree
151 print("-----")
152 DTacc = 100*accuracy_score(y_test,DTpred)
153 print("DT accuracy is: ", DTacc)
154
155 DTrecall=DTcm[1,1]/(DTcm[1,0]+DTcm[1,1])
156 print("DT Recall is: "+ str(DTrecall))
157 DTprecision= RFcm[1,1]/(DTcm[1,0]+DTcm[0,1])
158 print("DT Precision is: "+ str(DTprecision))
159 print("DT F-measure is: "+ str(2*((DTprecision*DTrecall)/(DTprecision+DTrecall))))
160
161 #for Naive Bayes
162 NBacc = 100*accuracy_score(y_test,NBpred)
163 print("-----")
164 print("NB accuracy is: ", NBacc)
165
166 NBrecall=NBcm[1,1]/(NBcm[1,0]+NBcm[1,1])
167 print("NB Recall is: "+ str(NBrecall))
168 NBprecision= NBcm[1,1]/(NBcm[1,0]+NBcm[0,1])
169 print("NB Precision is: "+ str(NBprecision))
170 print("NB F-measure is: "+ str(2*((NBprecision*NBrecall)/(NBprecision+NBrecall))))
171
172 #for Logistic Regression
173 print("-----")
174 LRacc = 100*accuracy_score(y_test,LRpred)
175 print("LR accuracy is: ", LRacc)
176
177 LRrecall=LRcm[1,1]/(LRcm[1,0]+LRcm[1,1])
178 print("LR Recall is: "+ str(LRrecall))
179 LRprecision= RFcm[1,1]/(LRcm[1,0]+LRcm[0,1])
180 print("LR Precision is: "+ str(LRprecision))
181 print("LR F-measure is: "+ str(2*((LRprecision*LRrecall)/(LRprecision+LRrecall))))
182
183 #for KNeighbors Algorithm
184 print("-----")
185 KNNacc = 100*accuracy_score(y_test,KNNpred)
186 print("KNN accuracy is:",KNNacc)
187
188 KNNrecall=KNNcm[1,1]/(KNNcm[1,0]+KNNcm[1,1])
189 print("KNN Recall is : "+ str(KNNrecall))
190 KNNprecision= KNNcm[1,1]/(KNNcm[1,0]+KNNcm[0,1])
191 print("KNN Precision is: "+ str(KNNprecision))
192 print("KNN F-measure is: "+ str(2*((KNNprecision*KNNrecall)/(KNNprecision+KNNrecall))))
193 print("-----")
194

```