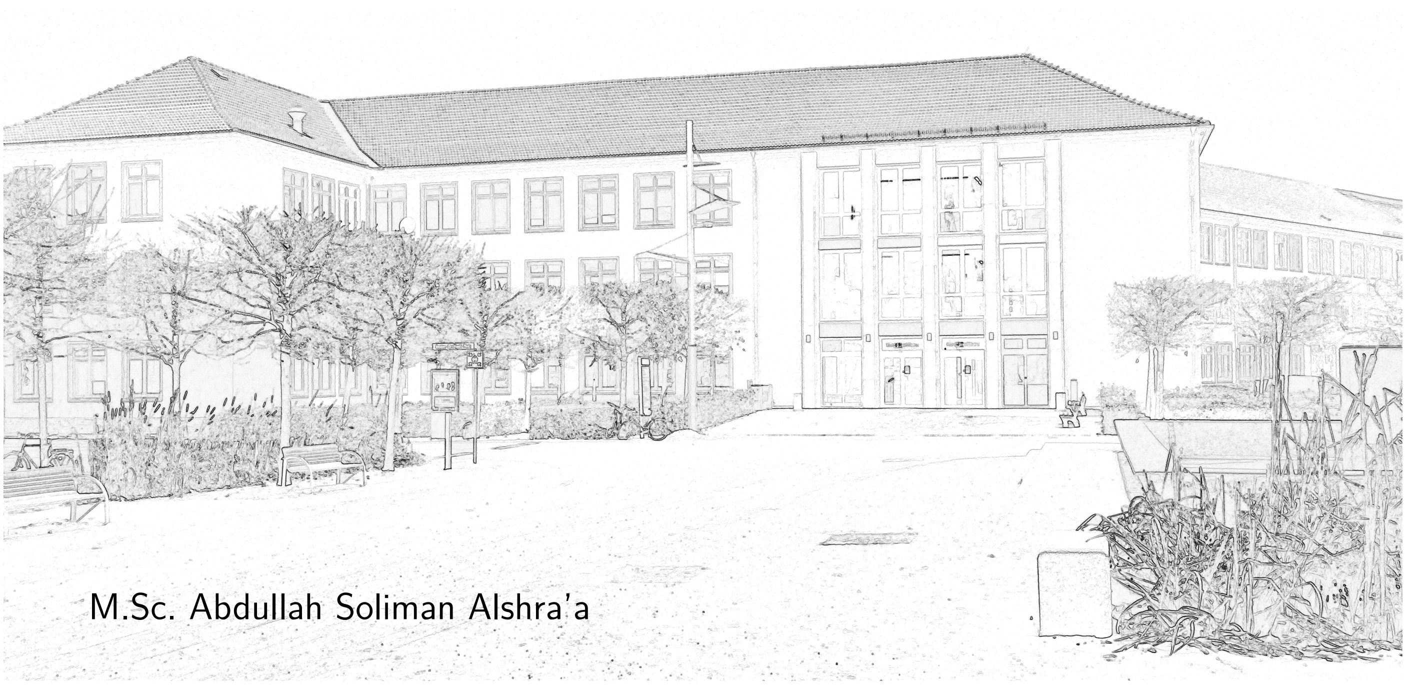
TECHNISCHE UNIVERSITÄT
ILMENAU

Intrusion Detection System against Denial of Service Attack in Software-Defined Networking



M.Sc. Abdullah Soliman Alshra'a

Ph.D. Dissertation in Electrical Engineering and Information Technology

Please cite as:
Abdullah Soliman Alshra'a, "Intrusion Detection System against Denial of Service Attack in Software-Defined Networking," Ph.D. Dissertation (Ph.D.arbeit), Technische Universität Ilmenau, Department of Electrical Engineering and Information Technology, January 2022. DOI: 10.22032/dbt.51449, URN: urn:nbn:de:gbv:ilm1-2022000015

Technische Universität Ilmenau
Department of Electrical Engineering and Information Technology
Communication Networks Group

Helmholtzplatz 2 · 98693 Ilmenau · Germany

http://www.tu-ilmenau.de/kn

Intrusion Detection System against Denial of Service Attack in Software-Defined Networking

Ph.D. Dissertation in Electrical Engineering and Information Technology

submitted by

M.Sc. Abdullah Soliman Alshra'a

born on 12. May 1985
in Jordan

in the

Communication Networks Group

Deptartment of Electrical Engineering
and Information Technology
Technische Universität Ilmenau

Reviewers:	**Prof. Dr. rer. nat. Jochen Seitz**
	Prof. Dr. Ing. Giovanni Del Galdo
	Prof. Dr. Ing. Jürgen Schussmann
Submission Date:	**27 August 2021**
Defense Date:	**27 January 2022**

DOI: 10.22032/dbt.51449
URN: urn:nbn:de:gbv:ilm1-2022000015

Acknowledgement

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Jochen Seitz for his guidance throughout the Ph.D. journey from the title's selection to completing the dissertation writing. His immense knowledge, motivation, and patience have given me more power and spirit to excel in all difficulties I have faced. Conducting the academic study regarding such a topic could not be as simple as he made this for me. He is my mentor and a better advisor for my doctorate studies beyond the imagination.

I should thank the Ilmenau University of Technology and Thuringian Graduate Funding Ordinance (ThürGFVO) for the Ph.D. scholarship that allowed me to fulfill this work. I would not forget to thank Prof. Michael Rock for his support from the point I applied for the scholarship until I presented my Ph.D. defense.

In addition, I would like to express my gratitude to the rest of the communication networks group, especially Ing. Michael Heubach and Dr. Maik Debes for giving encouragement and sharing insightful suggestions. They all have played a major role in polishing my research writing skills, therefore their endless support is hard to forget throughout my life.

I would always remember my fellow lab-mates for the fun time we have spent together, sleepless nights, and stimulating the discussions. I would also like to thank my friends from the university and all people I have met in Ilmenau city who made my residency comfortable and wonderful.

In the end, I am grateful to my father for his belief in me and my mother for her infinite love. I am also thankful to my brothers and sisters, friends, and acquaintances who remembered me in their prayers for the ultimate success. I consider myself nothing without them. They granted me enough moral support, encouragement, and motivation to accomplish this work. Many thanks to my lovely family, my wife Raghda, who shared with me the highs and lows of this journey, and my son Hashim whose existence and smile always inspire me and provide me the determination to realize my ambition.

Abstract

The exponential growth of online services and the data volume transferred over the communication networks raises the need to change the structure of traditional networks to a new paradigm that adapts to the development's demands. Software-Defined Networking (SDN) is an advanced network architecture aiming to evolve and transform the traditional network into a more flexible network that responds to the new requirements. In contrast to the traditional network, SDN allows decoupling of the control and data planes functionalities to monitor, configure, and optimize network resources efficiently. It has a centralized controller with a global network view to manage its resources using programmable interfaces. The central control brings new security vulnerabilities and acts as a single point of failure, which the malicious user might exploit to disrupt the network functionality. Thus, the attacker launches massive traffic known as Distributed Denial of Service (DDoS) attack from the SDN infrastructure layer towards the controller. This DDoS attack leads to saturation of control channel bandwidth and destroys the controller resources. Furthermore, the SDN architecture inherits some attacks types from the traditional networks. Therefore, the attacker forges the packets to appear benign and then targets the traditional DDoS objectives such as hosts, servers, applications, routers. This work observes the behavior of malicious users. It then presents an Intrusion Detection System (IDS) to safeguard the SDN environment against DDoS attacks. The IDS considers three approaches to obtain sufficient feedback about the ongoing traffic through the SDN architecture: the information from an external device, the OpenFlow channel, and the flow table. Therefore, the proposed IDS consists of three components; *Inspector* Device prevents the malicious users from launching the saturation attack towards the SDN controller. Convolutional Neural Network (CNN) Component employs the One-Dimensional Convolutional Neural Networks (1D-CNN) to analyze the controller's traffic through the OpenFlow Channel. The Deep Learning Algorithm (DLA) component employs Recurrent Neural Networks (RNN) to detect the inherited DDoS attacks. The IDS also supports distinguishing between malicious and benign users as a new countermeasure. At the end of this work, the network emulator Mininet and the programming language python model all the proposed components to test their feasibility. The simulation results demonstrate that the proposed IDS outperforms compared several benchmarking and state-of-the-art suggestions.

Kurzfassung

Das exponentielle Wachstum der Online-Dienste und des über die Kommunikationsnetze übertragenen Datenvolumens macht es erforderlich, die Struktur traditioneller Netzwerke durch ein neues Paradigma zu ersetzen, das sich den aktuellen Anforderungen anpasst. Software-Defined Networking (SDN) ist hierfür eine fortschrittliche Netzwerkarchitektur, die darauf abzielt, das traditionelle Netzwerk in ein flexibleres Netzwerk umzuwandeln, das sich an die wachsenden Anforderungen anpasst. Im Gegensatz zum traditionellen Netzwerk ermöglicht SDN die Entkopplung von Steuer- und Datenebene, um Netz werkressourcen effizient zu überwachen, zu konfigurieren und zu optimieren. Es verfügt über einen zentralisierten Controller mit einer globalen Netzwerksicht, der seine Ressourcen über programmierbare Schnittstellen verwaltet. Die zentrale Steuerung bringt jedoch neue Sicherheitsschwachstellen mit sich und fungiert als Single Point of Failure, den ein böswilliger Benutzer ausnutzen kann, um die normale Netzwerkfunktionalität zu stören. So startet der Angreifer einen massiven Datenverkehr, der als Distributed-Denial-of-Service Angriff (DDoS-Angriff) von der SDN-Infrastrukturebene in Richtung des Controllers bekannt ist. Dieser DDoS-Angriff führt zu einer Sättigung der Steuerkanal-Bandbreite und belegt die Ressourcen des Controllers. Darüber hinaus erbt die SDN-Architektur einige Angriffsarten aus den traditionellen Netzwerken. Der Angreifer fälscht beispielweise die Pakete, um gutartig zu erscheinen, und zielt dann auf die traditionellen DDoS-Ziele wie Hosts, Server, Anwendungen und Router ab. In dieser Arbeit wird das Verhalten von böswilligen Benutzern untersucht. Anschließend wird ein Intrusion Detection System (IDS) zum Schutz der SDN-Umgebung vor DDoS-Angriffen vorgestellt. Das IDS berücksichtigt dabei drei Ansätze, um ausreichendes Feedback über den laufenden Verkehr durch die SDN-Architektur zu erhalten: die Informationen von einem externen Gerät, den OpenFlow-Kanal und die Flow-Tabelle. Daher besteht das vorgeschlagene IDS aus drei Komponenten. Das Inspector Device verhindert, dass böswillige Benutzer einen Sättigungsangriff auf den SDN-Controller starten. Die Komponente Convolutional Neural Network (CNN) verwendet eindimensionale neuronale Faltungsnetzwerke (1D-CNN), um den Verkehr des Controllers über den OpenFlow-Kanal zu analysieren. Die Komponente Deep Learning Algorithm (DLA) verwendet Recurrent Neural Networks (RNN), um die vererbten DDoS-Angriffe zu erkennen. Sie unterstützt auch die Unterscheidung zwischen bösartigen und gutartigen Benutzern als neue Gegenmaßnahme. Am Ende dieser Arbeit werden alle vorgeschlagenen Komponenten mit dem Netzwerkemulator Mininet und der Programmiersprache Python modelliert, um ihre Machbarkeit zu testen. Die Simulationsergebnisse zeigen hierbei, dass das vorgeschlagene IDS im Vergleich zu mehreren Benchmarking- und State-of-the-Art-Vorschlägen überdurchschnittliche Leistungen erbringt.

Contents

Chapter 1

Introduction

A network consists of two or more computer systems that communicate to share resources. The network nodes are linked through cables, telephone lines, radio waves, satellites, infrared light beams, etc. Besides, the network nodes use standard communication protocols over digital interconnections to communicate with each other. The networking concept considers exchanging information and ideas among the network nodes with common objectives or particular interests.

Over the past three decades, networks faced increased traffic demands since the organizations and users gradually depend on network connectivity for trade, user service, communications, and resource sharing. That makes the traditional networks unsuitable for managing the massive amount of exchanging data. Therefore, Software-Defined Networking (SDN) confers reliable communication and emerges as a solution and evolutionary approach to the traditional network structure and the growing trend in networking.

This chapter illustrates the dissertation outlines and provides a preamble introduction to the SDN concept, followed by the problem addressed in this work. Moreover, the chapter highlights the objectives and the contribution of the dissertation. The chapter presents a brief introduction and the motivation in section 1.1. Afterward, it explains the problem statement in section 1.2. Section 1.3 lists the work objectives, and section 1.4 shows the contributions. Finally, section 1.5 presents the dissertation outline.

1.1 Motivation

Nowadays, the rapid increase in network traffic makes the communication networks more complex than the networks three decades ago. By 2023, around two-thirds of the world's population will have access to the internet. In addition, there will be 5.3 billion active internet users compared to 3.9 billion in 2018 [Cis20]. Moreover, the new technologies (e.g., Internet of Things (IoT), Machine Learning, and Deep Learning) could increase the number of connected devices to be more than 70 billion worldwide by 2025 [Sta19].

The new technologies trends require a faster response, higher security, and easier manageable networks adapting to the massive number of the connected user. However, the conventional networks would no longer be able to meet the growing requirements. Therefore, SDN appears as a new networking approach that can improve the network performance and meet all these changes. SDN enables network operators to quickly manage, configure and monitor their networks using programmable interfaces. It provides central control of the network, which offers more flexibility. Also, SDN minimizes the operating and maintaining costs and allows fine-grained traffic control or fast service deployment [AMK+18, GBCMVVLV20].

In conventional networks, the control and data plane are situated together as a single network entity (forwarding node). But, this forwarding node suffers from slow service deployments and high operating costs due to the increased number of connected devices. Therefore, SDN separates the controller plane from the data plane and presents a new network structure comprising three isolated layers. On top of the SDN structure, the application layer handles all the end-user activities and analyzes the network operation. the application layer contains the high network policies and applies its decisions through the underneath layers. In the middle of the SDN structure lies the control layer, which consists of SDN controllers devices. The controller receives instructions from the application layer, has a central view of the network, handles the ongoing traffics and manages the network. The controller has applications configured according to the administrator's policies. These applications aim to manage the network traffic and make a decision on behalf of the data plane in case a new flow reaches the switch with new features. On the bottom of the SDN structure, The infrastructure layer contains the forwarding nodes (e.g., routers, switches) and is responsible for forwarding and processing the ongoing packets according to the rules and policies decided by the control plane. Each switch has a flow table with the installed instructions. However, if the switch receives a packet that does not match any entry in the flow table, the switch uses a `Miss`

`Table` entry which forwards the received packet as *Packet_In* message to the controller. After that, the controller extracts the packet header and obtains the important features for the routing algorithm to respond with a *Packet_Out* message containing the proper rule to be installed in the requesting switch's flow table [KDA19].

1.2 Problem Statement

Southbound Application Interface (Southbound API) is a bidirectional interface located between the control and infrastructure layers. It allows the SDN controller to communicate with the SDN switches to provide essential information like the current states of links or switches. The most important task for the Southbound API is to transport the *Packet_In* messages from the SDN switch to the controller and the *Packet_Out* reversely. OpenFlow protocol is one of the first SDN standards that define the communication methods between the SDN controller and switches over Southbound API. OpenFlow protocol initially enables the SDN controller to directly interact with the forwarding devices such as switches and routers. It facilitates adding, modifying, and removing rules and instructions in the flow table. So, the OpenFlow protocol makes the network more dynamic and responsive to real-time changes and traffic demands.

Although the communication process between the SDN controller and the switches is effective and enhances the flexibility and programmability, it provides new vulnerabilities and security challenges. The SDN controller is the brain of the network, and all forwarding nodes depend on the SDN controller to manage the flow traffics. But, if the controller is out of service, the entire network would be out of service. This problem is known as a single point of failure.

Therefore, the adversary exploits this process and crafts a massive volume of packets with different features (e.g., IP and MAC addresses). The crafted packets do not match any installed flow entry when the adversary injects them through the switches. Hence, the *Miss Table* will forward the unmatched packets to the controller. This type of attack aims to consume control channel bandwidth and limited resources in both the control and switches. It is recognized as a type of Denial of Service (DoS) attacks known as the saturation or injection attack [DGLG17].

Moreover, SDN is susceptible to the inherited types of attacks from conventional networks. This way, the adversary exploits the SDN architecture and performs different malicious tasks. Also, the attack mainly uses the predefined flow rules in the flow table. The adversary crafts the packets to have the appearance of benign packets. Then, the attack utilizes the infrastructure layer to target the traditional DDoS objectives such as hosts, servers, applications, routers, etc [Bha19].

1.3 Objectives

As the title of the dissertation states and conforms to the described motivations and challenges in the previous sections. The main goal of this work is to develop an Intrusion Detection System (IDS) framework. The proposed IDS addresses the DDoS attack problem in the SDN environment in case of new vulnerabilities or inherited threats of the legacy network structure. In more detail, the dissertation aims to:

1. Investigate the possibilities to launch DDoS attacks and define the vulnerable aspects in the SDN structure.
2. Design an IDS framework dealing with all DDoS vulnerable aspects, whether new or inherited threats.
3. Investigate the SDN controller ability to perform under the conditions of DDoS attacks in presence of the proposed IDS component or without.
4. Define all available approaches to collect information about the traffic behavior in the infrastructure layer and use the SDN controller to analyze them.
5. Make the SDN controller more intelligent by providing methods that grant the SDN controller the ability to think and recognize new malicious attacks.
6. Determine the parameters that the controller can obtain from the network and assign the suitable parameters as an input to the models and algorithms that analyze the network behavior.
7. Enable the SDN switches to block DDoS attacks and recognize the benign packets to forward them only.

1.4 Contributions

Throughout the dissertation, the contributions that have been realized towards the objectives as mentioned earlier are briefly:

1. A review into SDN concept, its structure, associated problems, and the relevant work.

2. A thorough overview and in-depth discussion of the SDN vulnerabilities and the security threats caused by DDoS attacks.
3. The achievement of the IDS framework consisting of three components working individually to prevent the network's resource against DDoS attacks.
4. Evaluation of the proposed three components compared to the state-of-the-art.
5. The implementation of the Artificial Intelligence (AI) concept and statistical methods on the SDN controller and applying them to recognize the malicious behavior.
6. The suggestion toward the use of Internet Protocol Security (IPSec) to improve the OpenFlow switch performance to differentiate between the benign and malicious traffic flow.

1.5 Outline

This dissertation is organized as the following:

Chapter 2 provides a background of the SDN principle. It introduces an overview of the SDN layers and the existing APIs. Moreover, chapter 2 presents the OpenFlow protocol, its versions, and the exchange messages. Then, chapter 2 demonstrates the architecture of the OpenFlow switches and SDN controller.

Chapter 3 highlights the threats that the SDN structure encounters, introduces the problem statement, and classifies the DDoS according to the SDN environment. After that, chapter 3 shows the advantages that SDN offers to detect and prevent DDoS attacks and also provides insight into the state-of-the-art solutions to address DDoS attacks.

In chapter 4, the proposed IDS framework is clarified with a detailed description of its components and their functionalities. Besides, chapter 4 shows an overview to apply the AI concepts and explain how the proposed IDS framework adopts them. A discussion at the end of this chapter presents the essential ideas and various obstacles.

The simulation environments, network typologies, and evaluations are presented in chapter 5. Additionally, chapter 5 provides and discusses details on the simulated scenarios, AI models, and the obtained results. A brief discussion follows at the end of the chapter. Finally, chapter 6 concludes the work contribution and addresses some open issues and challenges for future works.

Chapter 2

Principles of Software-Defined Networking

Computer networks ordinarily aim to share information between all connected devices from one endpoint to another. Internet acts as a great network example where users, business foundations, industries, and other units exchange information to realize their objectives. New complex issues invite the network administrations to modify the network configurations dynamically because of the evolution in the computer systems and the increasing of connected networks, .

The computer networks' architecture needs forwarding devices, which is responsible for transferring data under various standard protocols. The forwarding devices mainly comprise data and control layers, as depicted in Figure 2.1. The control layer has a CPU, memory routing table, and routing algorithms. The routing table (or more than one routing table) contains routing information known as flow rules. The routing table is calculated by routing algorithms or the developer's configuration (connected routes, statistical routes). Consequently, there could be more than one applicable rule for the same flow. But, the best flow rule is installed in the forwarding table.

The infrastructure layer, called the forwarding or data layer, contains the input port (Inport), output ports (Outport), the forwarding table, and the switch fabric. The forwarding table has the definitive destination rule that routes the packet to given IP prefix (or MAC address depending on the layer). When the Inport receives the packets, it looks up the forwarding table for the applicable flow rules. Then, the switch fabric forwards the packet to the proper Outport, which transmits the packet through the link to the next hop. However, suppose there is no matching flow rule in the forwarding

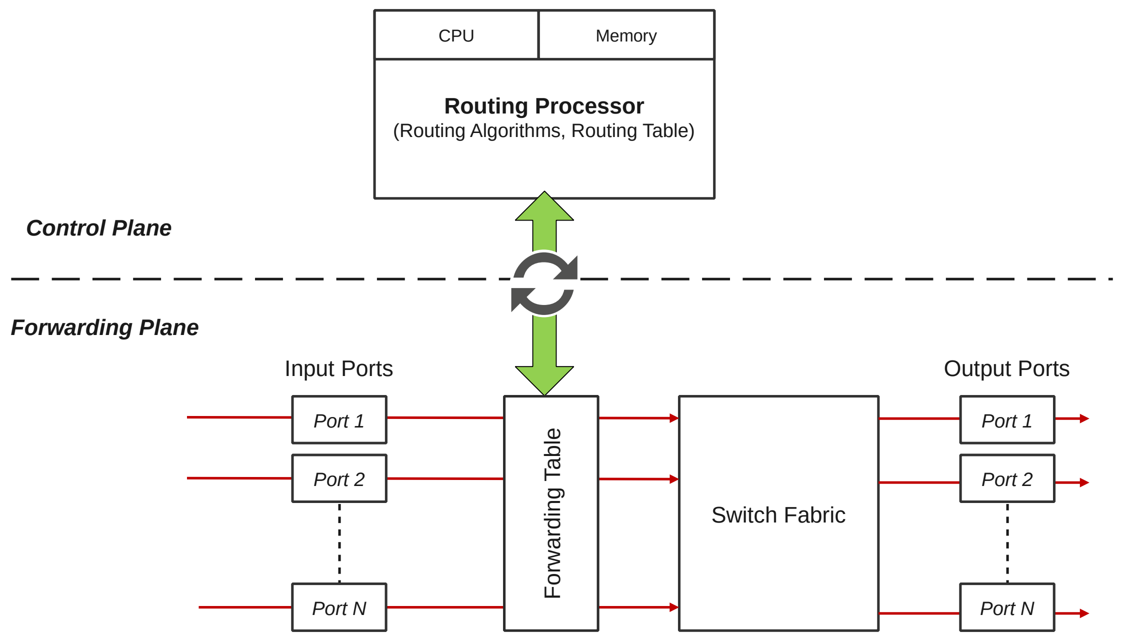


Figure 2.1. – Basic Router Architecture

table. In that case, the default flow rule pushes the packet through other functions based on the routing algorithms (e.g., packet filtering) [MR17] [ASS18].

In such a legacy network, the infrastructure and control layers are combined in a single physical device (forwarding device) as a standalone entity. This architecture brings various challenges and drawbacks to the network administrators with increasing nodes' number, adding a new network policy, incorporating network rules, or configuring network policies. In the first Challenge, the forwarding nodes must be configured individually for the entire network, which means more processing time and operating costs [WL20]. Secondly, the vendors are forced to discover the internal work of their network devices, such as routers, switches, and wireless access points. Thirdly, the network scalability has to respond to the changing traffic demands. Last but not least, there is a need for a network that has a holistic view of all the network resources due to the new service-oriented requirements [EJNJ21].

Traditional network protocols have been designed to work in a distributed control environment. Each forwarding device transfers the data based on predefined algorithms. However, the traditional networks are not flexible enough because the programming task is challenging in such human-dependent architecture and distributed control environment by multi-vendors and different protocols. This way, the developers should specify the low-level operations of the routing algorithms in detail. [MBES16].

Based on the mentioned earlier, SDN solves the complexity problem by making the network configuration more controllable, manageable, and centralized. SDN comes with a new structure that is appropriate for today's applications to perform well [HM21]. This chapter firstly indicates an overview of the SDN paradigm and introduces the general SDN structure. Moreover, the chapter illustrates the OpenFlow protocol in terms of its role and different versions. Afterward, the chapter explains the architecture of both the OpenFlow switch and the SDN controller.

2.1 Overview of SDN Architecture

According to the Open Network Foundation (ONF), [ONF] The primary motivation of SDN is to solve problems of the traditional networks and to accelerate innovation. SDN physically dissociates the network control layer from the infrastructure layer, which manages several forwarding devices by a single control layer. Consequently, SDN presents a different architecture, makes the network devices manageable, programmable, cost-effective, adaptable, dynamic, and suitable for high bandwidth, and enables today's applications to perform well. Moreover, the SDN concept supplies virtualized networks with a different approach in network design and management.

The SDN architecture has physical devices that are just forwarding elements without control functions in the infrastructure layer. All forwarding devices are dumb and execute predefined instructions installed by the intelligent control, which is removed from the forwarding layer to a separated control layer. Thus, the isolation of the control layer promotes the network administrators and vendors to evaluate, debug, and test the new SDN design before deploying it in the real network.

In other words, the SDN framework contains three layers (layers): the application layer, control layer, and forwarding layer, as shown in Figure 2.2. The application layer is located at the top of the SDN architecture, where specific applications handle the abstract policy of the network and end-user activities. Also, the application layer interacts with the control layer to configure the network services. The control layer represents the intelligence part of the network because it contains the network operating

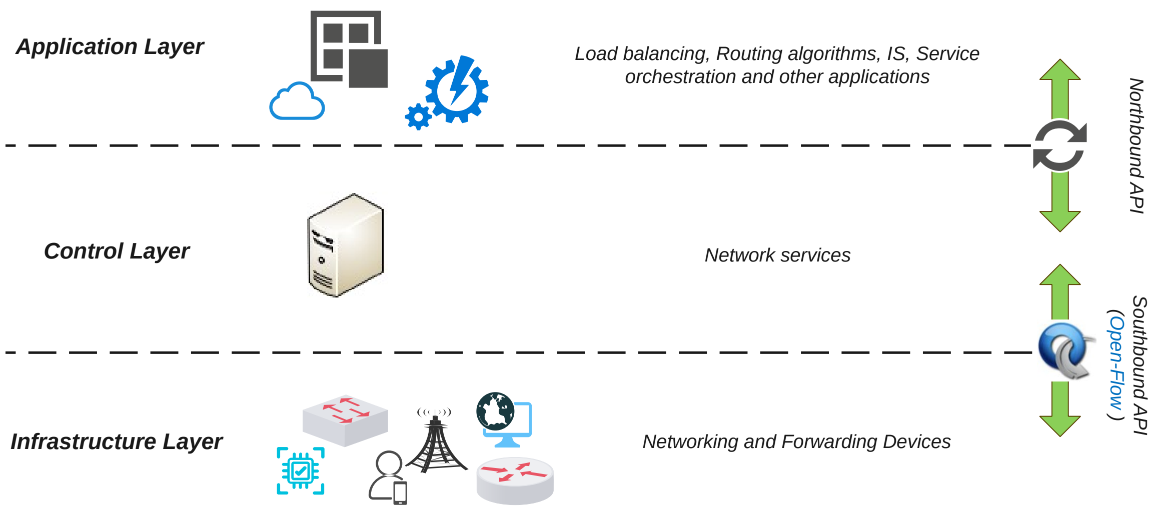

Application Layer
Load balancing, Routing algorithms, IS, Service orchestration and other applications
Northbound API
Control Layer
Network services
Southbound API
(Open-Flow)
Infrastructure Layer
Networking and Forwarding Devices

Figure 2.2. – SDN Architecture.

system (the SDN controller) and has a comprehensive view of the network infrastructure (forwarding layer). At the bottom of the SDN architecture, the infrastructure layer includes dumb devices that execute actions installed by the control layer. The following subsections introduce more details of these layers.

2.1.1 Infrastructure Layer

The infrastructure layer carries data traffic from one point to another within the network by buffering, scheduling, and forwarding the data. The infrastructure layer involves forwarding elements that receive the incoming data and transmit it through the appropriate Outport according to the forwarding table entries (flow table).

As Figure 2.3 shows. Once the packet reaches the forwarding node, the forwarding node looks up whether its forwarding table has the same data attributes (packet header) or not. The `Miss Table` flow specifies how to process packets that do not match any flow rule in the flow table. Therefore, in case there is no matching entry in the flow table,

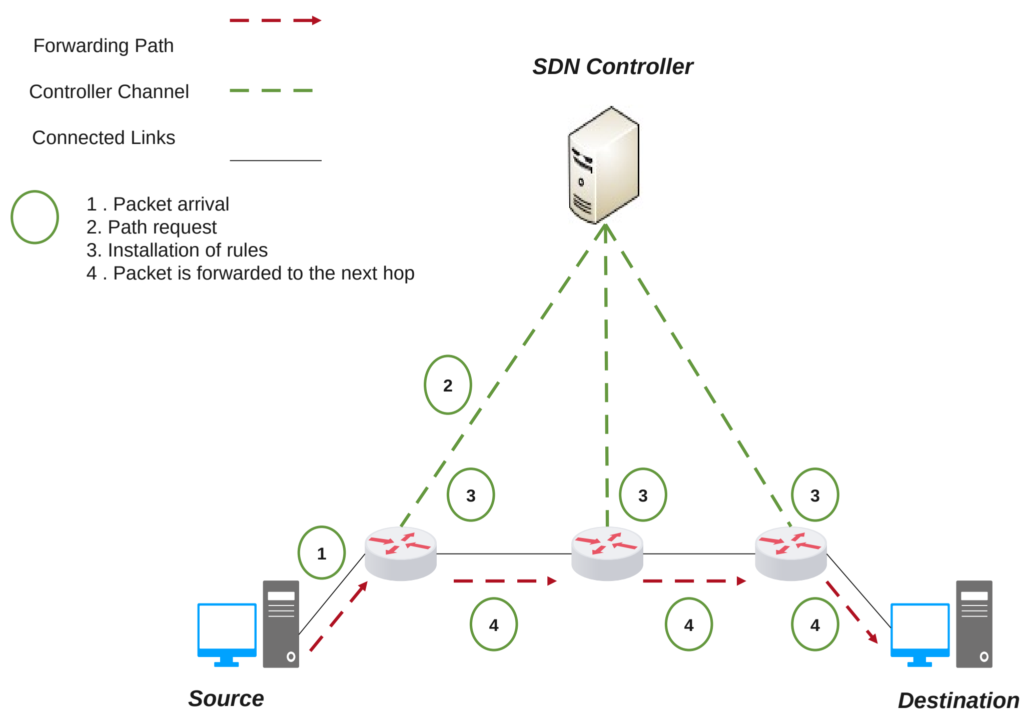


Figure 2.3. – Flow Management

the `Miss Table` flow forwards the packet or part of the packet to the control layer. Then, the control layer decides the appropriate conduct and responds by installing a new flow rule. Eventually, the forwarding node forwards or drops the packet based on the new flow rule. However, there are currently two available types of SDN switches. Firstly, the physical switch is a hardware device used to connect devices and enable communication across a network such as Pronto 3290 and 3780, IBM Rack, G8264, and Juniper MX-Series. Secondly, the virtual switch, a software application, allows virtual machines to connect to the outside world, such as Open vSwitch, Pica 8, Nettle, Panton, Indigo, etc [XWX20, YY15].

2.1.2 Control Layer

The control layer is the brain of the SDN network. It acts as a bridge between the network administration and the forwarding devices. Consequently, the control layer is responsible for configuring the forwarding devices according to the potential network policies in the application layer (e.g., routing, security, etc.). That means the control layer controls how the data packets are forwarded from one point to another by creating flow rules and installing them into the forwarding device.

The intelligence of the control layer is provided by centralized SDN controller software located on a server. The SDN controller plays this layer's outstanding role because of its global view and complete network knowledge. In other words, the developers and researchers can install the network policies and their innovations into the infrastructure layer through the control layer, which plays as an intermediate layer between them [MBES16].

Controller	Programming Language	Further development
Ryu	Python	Yes
POX	Python	No
Floodlight	Java	Yes
OpenDaylight	Java	Yes
Beacon	Java	No
ONOS	Java	Yes
NOX	C ++, Python	No
Trema	Ruby, C	Yes

Table 2.1. – SDN Controllers

On the one side, various SDN controllers have been presented. They are often open-source and under evolution. As Table 2.1 summarizes, which introduces brief information about the best-known SDN controllers [MBES16]. In general, the developers consider several factors in selecting an appropriate platform, such as programming language, cost, complexity, the available options to scale and accommodate the network requirements, etc.

Although all controller types have the same working concept, they differ in handling the forwarding requests due to the various applied routing algorithms. For example, when the OpenFlow switch receives a new packet that does not match any flow rule, the `Miss Table` entry forwards the packet towards the control layer. Then, the SDN controller inspects the sensitive fields (e.g., source addresses or destination addresses) and uses them to make a decision [XWX20].

On the other side, the control layer could include one or many controllers. In the one controller case (centralized architecture), the SDN controller manages all the infrastructure requests overloading the controller resources, particularly the vast networks (e.g., enterprise or data center networks). Furthermore, the one controller located in a single server, a physically centralized method representing a single point of failure and a potential bottleneck. Thus, the single controller is not always appropriate for networks due to a lack of reliability, scalability, security, performance, and availability.

As Figure 2.4 illustrates, a logically centralized control layer provides an alternative structure. It physically includes distributed controller servers connects to the forwarding devices through the Southbound interface. Thus, the forwarding devices see a single control server as a centralized entity. As a sequence, several controllers might work in flat architecture and perform the same tasks for either the whole network or a certain domain. In other cases, the controllers may work together in a hierarchical architecture to distribute the tasks and tune the controller performance [AR18].

2.1.3 Application layer

The application layer is located (also called the management layer) on the top of the SDN structure. It contains network applications drawing the network features (e.g., network management, traffic engineering, network management, load balancing for ap-

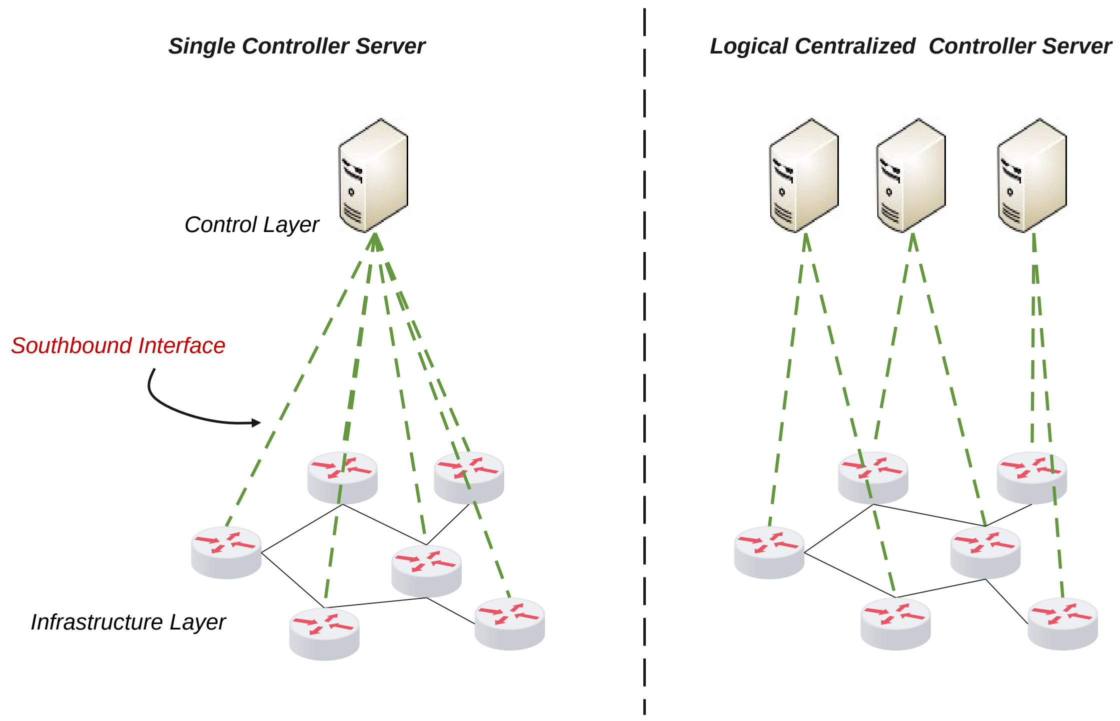


Figure 2.4. – Logical Centralized Control Layer

plication servers, security, network access control, network virtualization, etc [SKBJ20]). Furthermore, the application layer smoothly allows the developers to apply their ideas and innovation. In this layer, the network applications are written by particular programming languages. Moreover, specific compilers prepare (translate) the written application to the correct Northbound interface. In fact, using this layer is not common because most applications are written directly on the controller [MBES16].

The application layer receives feedback about the network status from the SDN controllers. Therefore, the control layer retrieves all needed information from the infrastructure layer. Hence, the administration uses the received information to provide the appropriate guidance to the control layer. Consequently, all the network information must be available for the application layer to offer and enhance different network applications (e.g., network automation, network configuration and management, network monitoring, network troubleshooting, network policies, and security).

A particular device achieves a specific objective in traditional networks, such as a firewall or load balancer. Nevertheless, the SDN concept replaces these devices with an application implemented in the control layer. Hence, the controller directly manages the infrastructure layer. Recently, many applications have been evolving, and there is initiative to have an application store support. The store would be under an SDN environment, where the concerns might install and download the desired application immediately [SKBJ20].

2.1.4 SDN Application Programming Interfaces

One of the essential objectives of SDN is to create centralized management that offers the requested policies to all appliances in the infrastructure layer. Hence, the vendors do not discover the product's specifications. Therefore, Application Programming Interfaces (APIs) enable the SDN architecture layers to be connected, as Figure 2.5 shows, even though the network architecture is physically isolated.

There are mainly three APIs types in the SDN environment that ensure a full abstracted communication as the following [LSL$^+$20]:

1. **Southbound Application Programming Interface**: OpenFlow and other protocols are proposed and used as Southbound APIs. These protocols transport the configurations from the controllers to the forwarding devices such as switches and routers. Southbound API is located between the control layer and the infrastructure layer devices. Thus, this API facilitates the communication between the controller and the forwarding devices. Also, the SDN controller adds, deletes,

and modifies flow entries in the forwarding devices according to the predefined algorithms.

OpenFlow protocol is deemed as a standard Southbound API and is typically the most used API. The other proposals are often extensions to provide configuration capabilities or dependent upon OpenFlow [LSL+20]. Generally, the Southbound interface enables programmability and rapid reconfiguration. Thus, the controllers can respond to the network changes and modify the network instruction at any time. Furthermore, the controller uses Southbound APIs to retrieve all available statistics regarding the network infrastructure states such as devices, ports, and links.

The Southbound API transports notifications provided by the SDN controller to the forwarding devices. Reversely, the Southbound API also transports information from the forwarding devices to the SDN controller. Eventually, Transport Layer Security (TLS) is applied to secure the communication between controller and

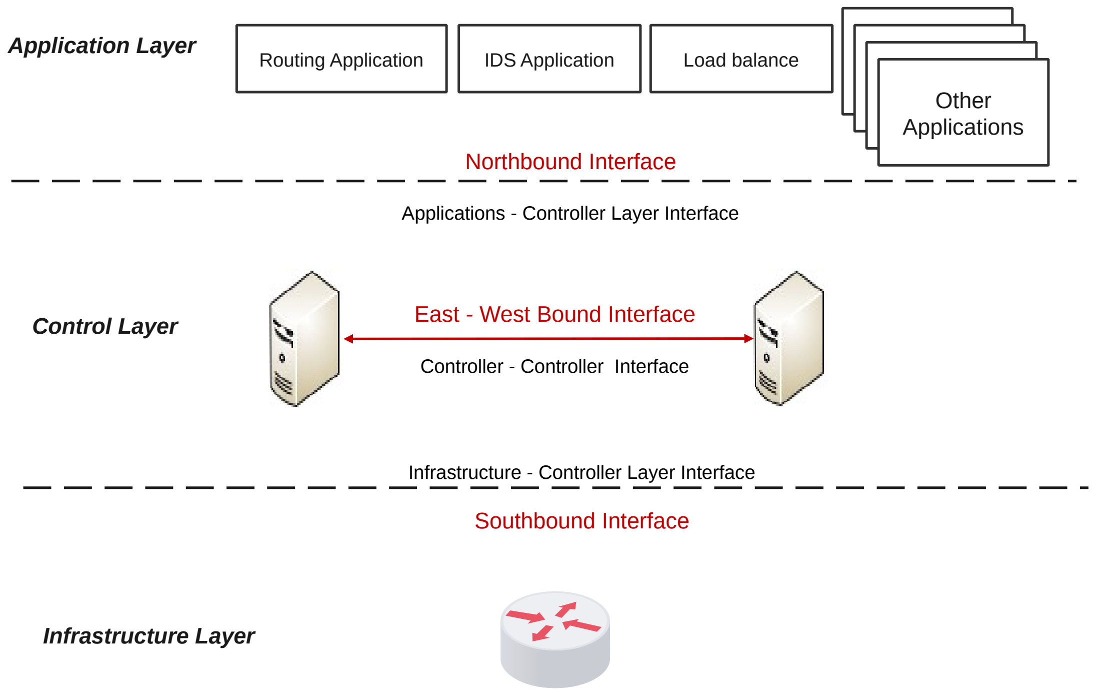


Figure 2.5. – SDN Application Programming Interfaces

switch. TLS is a technique to encrypt the communication between the controller and switch using public and secret keys as is described in [LSL$^+$20].

2. **East-West bound API**: East-West bound API is the physically distributed control layer interface that allows data exchange between the SDN controllers. Due to the presence of distributed controllers, the SDN controllers sometimes need to exchange information about the inter-domain or intra-domain situation. Consequently, the SDN controllers obtain the network state and bring scalability, deployability, and interoperability to the control layer. In other words, East-West bound is essential to achieving the benefits of the logically centralized controller. However, East-West API is still considered an open research area, with no standard East-West interface [BBMB16]. Many works proposed different interfaces, such as the Software-Defined Networking interface (SDNi). SDNi is an East-West protocol that contains different types of messages, e.g., Flow setup/tear-down/update and Reachability update [YXT$^+$12].

3. **Northbound API**: The main goal of Northbound API is to apply the ideas, innovations of the developers or the administrator desired instructions on the network infrastructure. Consequently, the Northbound API is the interface between the control layer and the application layer. The Northbound API interacts with the control layer devices once the applications need. For example, the applications measure the Quality of Service (QoS), install a load-balancing solution, force specific security settings, etc. However, that is not as easy as it should be because of the lack of standardization of Northbound API proposals. In contrast to the Southbound API, the Northbound API has no independent protocol because the Northbound API has not been defined until now. But, the ONF has started a Northbound Interface Working Group recently [HCA$^+$20]. Currently, the Northbound API is mostly software or programs. Therefore the SDN controllers often use the Representational State Transfer (REST) API as the Northbound API that provides a standardized interface for application development. Also, some SDN controllers suggest their high-level interfaces referred to controller-based APIs and other controllers use ad-hoc APIs [LSL$^+$20].

2.1.5 Benefits of Software-Defined Networking

Recently, the interest in SDN increases and many methods have been presented to realize an optimal structure. Also, different SDN technologies appear to support mobile, enterprise, or data center networks. SDN becomes more popular, where the research community pays attention to the potential benefits obtained from the SDN. In reality, the potential benefits of SDN are proved by granting a proper phase, particularly for experimenting with new suggestions and boosting novel system plans. In this subsection, the following points present the main benefits of SDN:

- SDN minimizes the network overhead since there is no need to control the network traffic from forwarding device to forwarding device. Also, the change of the network configuration is smoothly done over the centralized network controller [BGB18].
- The development of QoS is easier because of the central management. The SDN controller is responsible for directing the data traffic and hence guarantees the data delivery [BGB18].
- SDN is scalable and accepts new innovated units. So, the developers can apply their ideas without needing to learn the complex features of the potential devices [BGB18].
- SDN significantly minimizes and ends the downtime errors problem. The downtown error comes during the manual interruption (human intervention) to configure the individual networking devices. Consequently, the SDN maximizes the forwarding device up-time [BGB18].
- SDN allows the administrator to define security network policies centrally through the controller and harmoniously installs them into the whole network [BGB18].
- SDN reduces the operating costs for enterprises because the new policies or services can be seamlessly integrated to all switches by the central controller. Thus, saving time for individual node configurations. In addition, the SDN allows using virtual resources and can reduce the need for traditional propriety hardware. Finally, due to the central management, all the services might be monitored and managed, which also saves administrative costs [AMK$^+$18].
- Vendors do not need to expose the features of their devices [BGB18]. Furthermore, the SDN protocols are pure and independent software. Therefore, the developer can use them with different appliances instead of using over-featured appliances running proprietary protocols [GBC16].
- Internet expansion shows a significant challenge. For example, increasing the number of intelligent devices, sharing resources and massive amounts of the manipulated data that are stored on different methods (e.g., data center and clouds). Thus, the network operators, administrators, or service providers have to

manage all the stored information. Readily, the SDN enables management and control of the internet storage devices [GBC16].

- SDN provides the control of the entire network, which smooths the management and configuration of networking resources. While the number of network nodes grows every day, the separation between the control and infrastructure layer provides efficient control over the network traffic and resources allocation. For businesses that require integrating various services or applications, the SDN allows a central point of control for all appliances in the network. For example, the centralized SDN architecture supports an enterprise that has to host a new application on many virtual machines together [AMK+18].
- Due to the existence of a single centralized point, SDN provides an efficient environment for programmers using automation tools that orchestrate and automate the SDN network. Using API such as REST API provides a common interface to program and develop applications [GBC16].
- Running the SDN controller as a software program enables the controller to tune its resource usage based on the necessary capability easily. Also, the SDN controller can respond to any emergency case rapidly. As well, the SDN controller can allocate its resources to increase the capabilities [HM21].

2.2 OpenFlow Protocol

As mentioned in subsection 2.1.4, the controller interacts with the infrastructure layer devices using the Southbound API, which diverse proposed protocols have been introduced. However, the available proposed protocols are mostly extensions to the OpenFlow protocol, the commonly used protocol in SDN. Thereby, the SDN controller installs the forwarding rules into the OpenFlow switch to control their behavior with all common traffic types [GBC16].

According to the OpenFlow specifications, Figure 2.6 shows the OpenFlow protocol as a middle-ware between the SDN controller and the OpenFlow switch [Oped]. Thus, the SDN controller uses a secure channel and adds, edits, and removes any flow entry in the flow tables with a proactive or reactive approach. Moreover, the controller can import different statistics about the network behavior. That gives a complete overview of the network stats and makes the proper decisions.

Once a packet reaches the Inport of the switch, the lookup process starts to find the corresponding instruction in the flow table to execute. Nonetheless, in case no flow entry

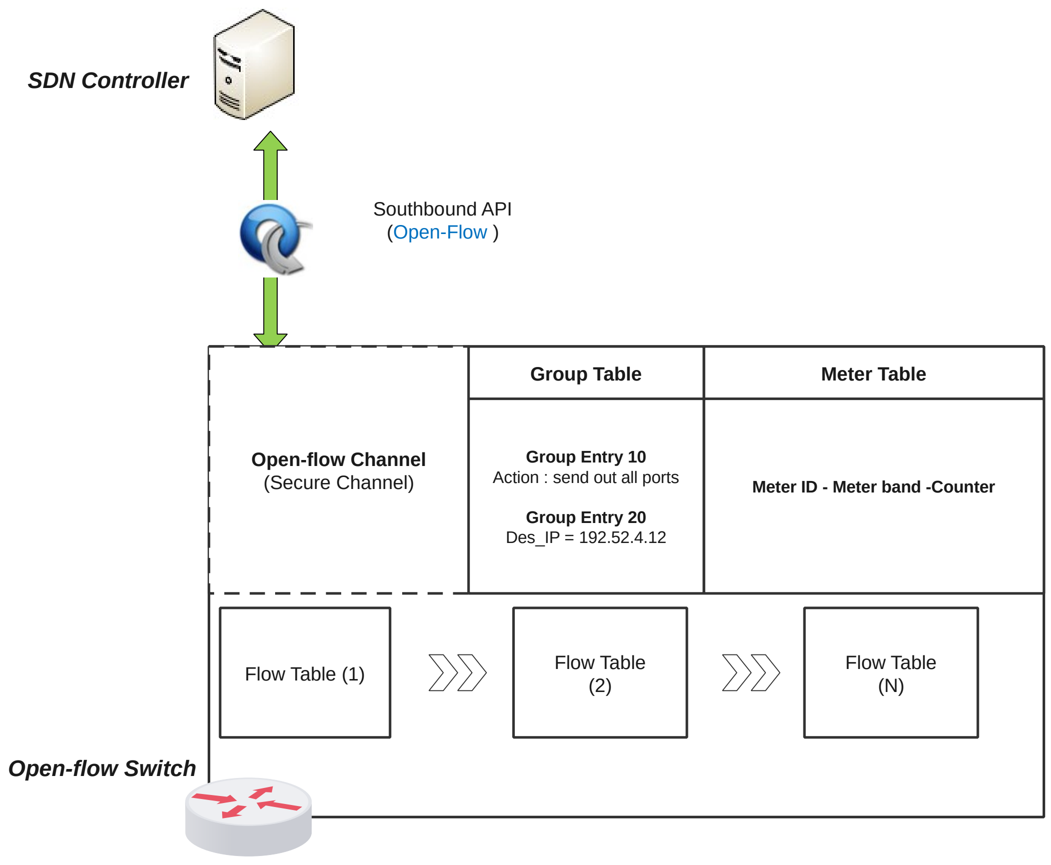


Figure 2.6. – OpenFlow Protocol Overview

matches the packet, the flow table has a `Miss Table` entry that forwards the packet through the OpenFlow channel to the SDN controller. The SDN controller decides the proper behavior to forward or drop the packet based on the predefined policies. Thereby, the SDN controller replies with instructions to the switch, and hence the packet is processed accordingly. The section presents a brief explanation of the OpenFlow protocol, controller-to-switch messages, and OpenFlow protocol versions [Oped].

2.2.1 OpenFlow Protocol Versions

OpenFlow protocol facilitates the programming of the forwarding devices to manage and direct the traffic according to the network policies. On the contrary, the vendors can offer many switch types with different programmability levels. That might restrict the administrator's ability to control the traffic. It could also introduce the inconsistent of traffic management between equipment from multiple vendors. Therefore, the OpenFlow protocol developers target to improve functionality and consistency by issuing advanced

OpenFlow versions. The table 2.2 shows a summarized comparison for different versions of the OpenFlow protocol [ONF] [CHL15].

Version	**Amendments**	**Target**	**Application**
1.1	Multiple table	Avoid flow entry explosion (Entry number is greater than the table capacity)	–
	Group table	Enable applying action sets to group of flows	Load balancing, failover, link aggregation
	Full Virtual Local Area Network (VLAN) and Multiprotocol Label Switching (MPLS) support	–	–
1.2	OpenFlow Extensible Match (OXM) match	Extend matching, flexibility	–
	Multiple controllers	High availability/load balancing/scalability	Controller failover, controller load balancing
1.3	Meter table	Add QoS	–
	Table miss entry	Provide flexibility	–
1.4	Synchronized Table	Better table scalability	MAC learning/forwarding
	Bundle	Better switch synchronization	Multiple switch configuration
1.5	Egress table	Enabling packet processing in Output port	–
	Scheduled bundle	Better switch synchronization	–
2.0	Not formal yet	Flexible packets switching/ Programmable switch /Other proposals	

Table 2.2. – Features Comparison for Various OpenFlow Versions

The table 2.3 describes the components of the flow table entries structure in OpenFlow v1.0 [Opef]. The OpenFlow v1.0 is the first version published in December 2009 and is widely used. The OpenFlow v1.0 has a single flow table only. Each entry in the flow table has 12 options to match the header packet's fields (Ingress Port, Ethernet source, Ethernet destination, Ethernet type, VLAN ID, VLAN priority, IP source, IP destination, IP protocol, IP ToS bits, TCP/UDP source port, TCP/UDP destination port). For example, the entry could match the packet header based on source address, destination address, TCP/UDP source, and destination port numbers [BM14].

Header Fields	**Counters**	**Actions**

Table 2.3. – Entry structure in OpenFlow Version 1.0

- **Header Fields**: To match against packet.
- **Counters**: Counts of the packets that matched the flow entry.

- **Actions**: To apply on the matching Packet.

However, OpenFlow v1.1 was released in February 2011, which allows multiple flow tables and the group table. In addition, matching packets in OpenFlow v1.1 is supporting Multiprotocol Label Switching (MPLS) [Opeb]. After that, OpenFlow v1.2 was released in December 2011. In this version, the flow entry can match the packet header of ICMPv6 and IPv6: source, destination by OpenFlow Extensible Match (OXM) match. Also, OpenFlow v1.2 introduces the ability to modify the roles of the controllers (multiple controllers as Master or Slave) and flow labels. As a consequence, the entry structure is extended to include the new additions as shown in the table 2.4 [Opec].

Header Fields	**Counters**	**Instructions**

Table 2.4. – Entry Structure in OpenFlow Version 1.2

- **Header Fields**: To match against packets.
- **Counters**: Counts of the packets that matched the flow entry.
- **Instructions**: To modify the action set or pipeline processing.

In April 2012, OpenFlow v1.3 appeared with an extension of the QoS feature. Besides, the meter tables have been the most important addition in OpenFlow v1.3. Furthermore, the concept of the `Table Miss` entry as an extension for the flow table has been a second major improvement, in addition, to support the IPv6 Extension Headers as a matching field. Therefore the entry structure was upgraded as Table 2.5 [Oped].

Match Fields	**Priority**	**Counters**	**Instructions**	**Timeouts**	**Cookie**

Table 2.5. – Entry Structure in OpenFlow Version 1.3

- **Match Fields**: To match rules of the flow entry.
- **Priority**: To match precedence of the flow entry.
- **Counters**: Counts of the packets that matched the flow entry.
- **Instructions**: To modify the action set or pipeline processing.
- **Timeouts**: The maximum amount of idle time (if no packet matched during idle time, remove the entry) or hard time (if the hard time timeout is exceeded, remove the entry) for the flow entry.
- **Cookie**: Flow entry identifier specified by the controller.

In August 2013, OpenFlow v1.4 came with more attributes such as the Synchronized table that improves the table scalability and the Bundle to release the switches synchronization, especially in the configuration of multiple switches. Also, OpenFlow v1.4 added the ability for flow monitoring and assigned the 6653 as the default OpenFlow channel port [Opee]. Nevertheless, the next version, OpenFlow v1.5 (released in 2015), enables the switches to process the packets on the Output port and enhances the synchronization of the switches [Opea]. In both versions, 1.4 and 1.5, the entry structure has the same fields to accommodate the new additions, as is clarified in Table 2.6.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Table 2.6. – Entry Structure in OpenFlow Version 1.4 and 1.5

- **Match Fields**: To match rules of the flow entry.
- **Priority**: To match precedence of the flow entry.
- **Counters**: Counts of the packets that matched the flow entry.
- **Instructions**: To modify the action set or pipeline processing.
- **Timeouts**: The maximum amount of idle time (if no packet matched during idle time, remove the entry) or hard time (if the hard time timeout is exceeded, remove the entry) for the flow entry.
- **Cookie**: Flow entry identifier specified by the controller.
- **Flags**: Flags alter the way flow entries are managed.

2.2.2 OpenFlow Messages

The SDN controller and OpenFlow switches exchange different types of messages through the OpenFlow interface. That enables the controller to configure the switches, receive events from the switch, install the instructions to the switch *Packet_Out*, modify the flows, monitor, and manage the network resources. Generally, the OpenFlow protocol supports three main message types: controller-to-switch, asynchronous, and symmetric. Each message type involves several sub-types [Oped], which are further explained the essential OpenFlow Messages as the following:

- **Controller-to-switch messages** are initiated by the controller and used to request statistics directly and also configure the switch to execute various other tasks.

  - Features Message: This message reorganizes the identity and the basic features of the attached switch. Once the OpenFlow channel is set up, the controller transmits the OFPT_FEATURES_REQUEST message to the OpenFlow switch. Consequently, the switch responds with the OFPT_FEATURES_REPLY message containing information about the switch's identity and different statistics.
  - Modify-State Message: The SDN controller is responsible for the flow table entries in the switches. The Modify-State Message is used when the flow table entries must be installed, modified, or deleted. Also, the Modify-State Message modifies the switch port characteristics.
  - Read-State Message: This message is sent to the switch to obtain the required information about the current state, statistics, and configuration. For instance, OFPMP_FLOW requests information about flow entries. Also, OFPMP_-PORT_STATS obtains information about the switch ports statistics.
  - Packet_Out or Flow-Mod Message: When the controller receives new request (*Packet_In* message), the controller replies with a *Packet_Out* message, which contains the required instruction to be implemented on the packet.
  - Barrier Message: Sometimes, continuous operations reach the switch queues with different priorities. Therefore, the controller issued the Barrier Request message to request the OpenFlow switch to process all messages sent before the Barrier Request message. After that, the OpenFlow switch will process any messages sent after the Barrier Request message and notify about the operation's accomplishment.
  - Role-Request Message: A controller defines its authority on the switch using this message as a master or slave controller. Thus, the controller sets the instructions or requests information of the switch according to its role.

- **Asynchronous messages** are change-dependent messages. The switch submits the message based on a new network event, any change to the switch state, or in case of errors. The four types of asynchronous messages are described below.
  - *Packet_In* Message: The OpenFlow switch sends the *Packet_In* message when there is a mismatch in the flow table for the newly received packet. Therefore, the switch uses the `Miss Table` entry to transfer the task to the controller by forwarding the encapsulated packet or some header fields.

Furthermore, the developer can apply the same message as an action of any entry.

- Flow-Removed Message: The flow entries are removed from the flow table in two cases: (1) The predefined timeout is over. (2) The controller decided to remove the entry according to its algorithms. Therefore, the controller sends a Flow-Mod message to the switch, instructing the switch to delete the entry. Then, the switch replies with a Flow-Removed message to confirm the deletion of the entry from the flow table.
- Port-STATUS Message: Once the port is added, modified, or removed from the switch, the controller must be informed by the PORT-STATUS message, so the switch transmits the message to the controller.
- Error Message: The switch sends error messages to the controller with unexpected events or errors.

- **Symmetric messages** are sent either by the controller or the switch without any request needed.
  - Hello Message: This aims to establish a connection between the controller and the switch. Once the switch or the controller runs, the hello messages are transmitted to notify the second part about the used version. However, if the versions of both parts are not compatible, the Error message is transmitted with the type (Hello failed) and code incompatible.
  - Echo Message: After the connection is established correctly, both the controller and the switch periodically send Echo messages to inform the second part about latency, bandwidth, and liveliness.

2.3 OpenFlow Switch Architecture

The OpenFlow switch comprises various parts working to direct the traffic toward the destination. The OpenFlow specification defines the architecture of the OpenFlow switch implementation in four main blocks: the OpenFlow Channel (subsection 2.1.4), the flow tables, meter table, and the group table as depicted in Figure 2.6. According to the latest OpenFlow version, the components of flow tables, group tables, matching fields, and action fields are covered in this section [Opea].

2.3.1 The Pipeline Process

The Pipeline Process enables the OpenFlow switch to treat packets in many steps with different flow tables and grants the OpenFlow switch more control on flow traffic. The pipeline process is applied in two types of OpenFlow-compliant switches [AHMYR+17]. On the one hand, the **OpenFlow-hybrid** switch offers the pipelines for both the OpenFlow and the traditional Ethernet switching. The hybrid OpenFlow switch contains an OpenFlow instance and the traditional structure of the forwarding device. That allows the administrator to deploy SDN traffic steering using the OpenFlow protocol on the forwarding device infrastructure. Moreover, the OpenFlow-hybrid switch has the traditional Ethernet switching processing such as layer two switching, layer three routing, VLAN tagging, QoS, and ACL. On the other hand, the **OpenFlow-only** switch supports the OpenFlow pipeline only.

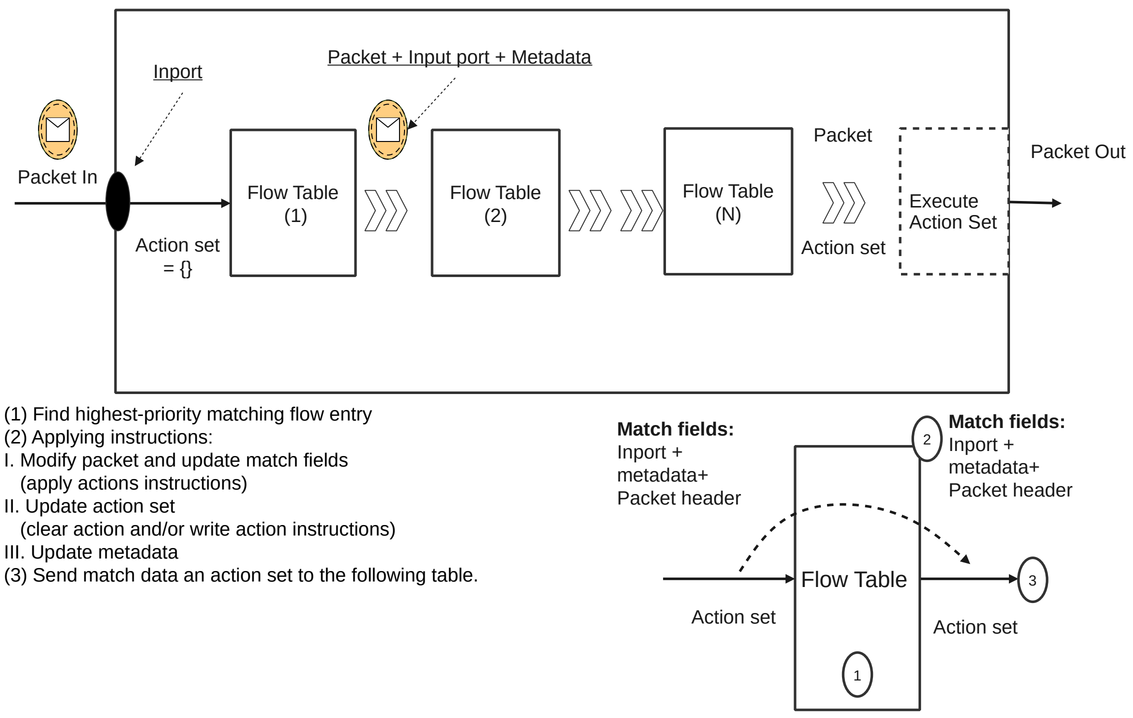

Inport
Packet + Input port + Metadata
Packet In
Action set = {}
Flow Table (1)
Flow Table (2)
Flow Table (N)
Packet
Action set
Execute Action Set
Packet Out
(1) Find highest-priority matching flow entry
(2) Applying instructions:
I. Modify packet and update match fields (apply actions instructions)
II. Update action set (clear action and/or write action instructions)
III. Update metadata
(3) Send match data an action set to the following table.
Match fields: Inport + metadata+ Packet header
Match fields: Inport + metadata+ Packet header
Flow Table
Action set
Action set
1
2
3

Figure 2.7. – Pipeline Process in OpenFlow Switch

Figure 2.7 demonstrates the pipeline processing sequence in the OpenFlow switch [Oped]. Each OpenFlow switch includes at least one flow table (see Figure 2.8), which keeps the installed instructions as entries. The switch uses the flow table to look up the packet information by comparing the packet attributes to the **Match** field of the entry.

Occasionally, the packet could match diverse match fields. The switch executes the action of the highest value in the **Priority** field. It is noteworthy that with updating the OpenFlow versions, the developers often try to update the available matched fields to be more flexible and suitable. That is evident in Table 2.7, which shows a sample comparison among the different OpenFlow versions.

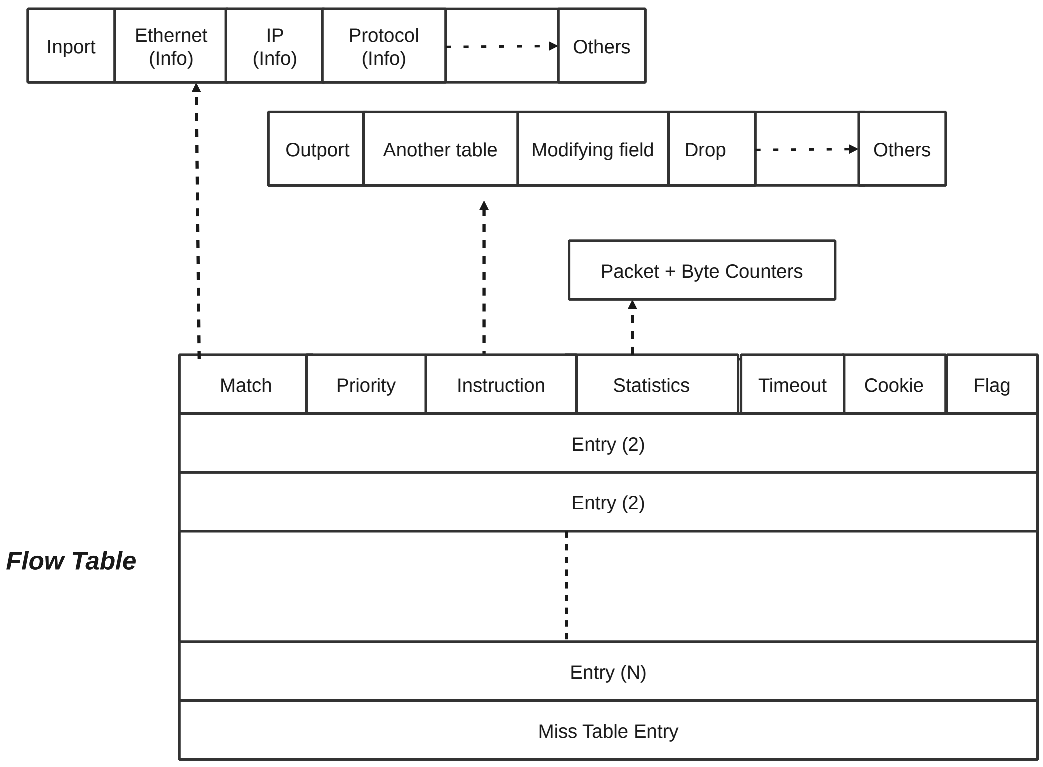


Figure 2.8. – OpenFlow Switch Table

Furthermore, every entry has a **Counter** field to count the number of packets that have used the flow entry during the past time. The counter increases when the packet executes the related action. The action is stored in the **instruction** field, which is responsible for handling the matched packet. Thus, instruction could modify the packet or change the related matched field or transmit the packet to another flow table for more processing.

As the section 2.2 mentions, the **Timeouts** field is used for OpenFlow versions 1.3 and up to define the maximum time before flow expiration by the switch. Also, the **Cookie** field is a value used by the controller to filter flow entries. In OpenFlow version 1.5, **Flags** manages the flow based on flag type, for example, the flag OFPFF_SEND_-FLOW_REM. If this flag is set, the switch must send a flow removed message to the controller. The message contains a complete description of the flow entry, such as the reason for removal, the flow entry duration at the time of removal, and the flow statistics.

Field	**1.0**	**1.1**	**1.2**	**1.3&1.4&1.5**
Ingress Port	✓	✓	✓	✓
Metadata		✓	✓	✓
Ethernet: src, dst, type	✓	✓	✓	✓
IPv4: src, dst, proto, ToS	✓	✓	✓	✓
TCP/UDP: src port, dst port	✓	✓	✓	✓
MPLS: label, traffic class		✓	✓	✓
OXM			✓	✓
IPv6			✓	✓
IPv6 Extension Headers				✓

Table 2.7. – Match Fields' Comparison for Various OpenFlow Versions

2.3.2 Group Tables

The flow entry is responsible for matching flows. It runs the proper action for relevant incoming packets on logical OpenFlow interfaces. However, the action sometimes works to direct the packets to a base action called a group action which executes a further process on the packets. In other words, the Group Table represents sets of actions for flooding and more complex forwarding semantics such as multi-path, fast reroute, and link aggregation. Each group table includes a set of entries, where the entry has a list of actions called buckets and a specific group type, as shown in the Table 2.8.

Group Identifier	**Group Type**	**Counters**	**Action Buckets**

Table 2.8. – Group Table Fields in OpenFlow Version 1.3

- **Group Identifier**: To uniquely identify the group.
- **Group Type**: To identify the type of the group as there are required group types and optional group types.
- **Counters**: Increased once a group processes a packet.

- **Action Buckets**: A list of actions is stored in an action bucket.

Furthermore, Group tables have been using in OpenFlow protocol from version 1.2, [Opec], and there are four types to execute the actions in the group tables:

1. **All**: To run all the action buckets in the group.
2. **Select**: To execute one action bucket in the group according to predefined roles or algorithms such as round-robin.
3. **Indirect**: To execute only one defined bucket in this group.
4. **Fast Failover**: To execute the first live bucket. Therefore, if the switch should forward a packet through port 1 but the port is down, the bucket uses the backup action to forward the packet through port 2.

2.3.3 Meter Table

The Meter Table has been used since OpenFlow protocol version 1.3 [Oped]. It includes different entries to determine a specific meter per flow. A meter entry enables QoS operations, such as rate-limiting, by monitoring the meter to a specific port queue to apply a complex QoS framework. The meter table fields are shown in 2.9.

Meter Identifier	Meter Bands	Counters

Table 2.9. – Meter Table Fields in OpenFlow Version 13

- **Meter Identifier**: To uniquely identify the Meter.
- **Meter Bands** : An unordered list of meter bands, where each meter band defines the rate of the band and the method to treat the packet.
- **Counters**: Increased by one when a packet is processed by a Meter.

2.4 SDN Controller Architecture

Figure 2.9 depicts the basic architecture of the SDN controller, where the controller usually has a set of pluggable applications or algorithms that work to achieve diverse network tasks such as routing, security, load balancing, etc. In general, the routing task is often the most important module because it is responsible for transferring the data in the network. Therefore, all the basic controller architecture contains this module. Furthermore, the controller architecture is developed by enhancing the exciting

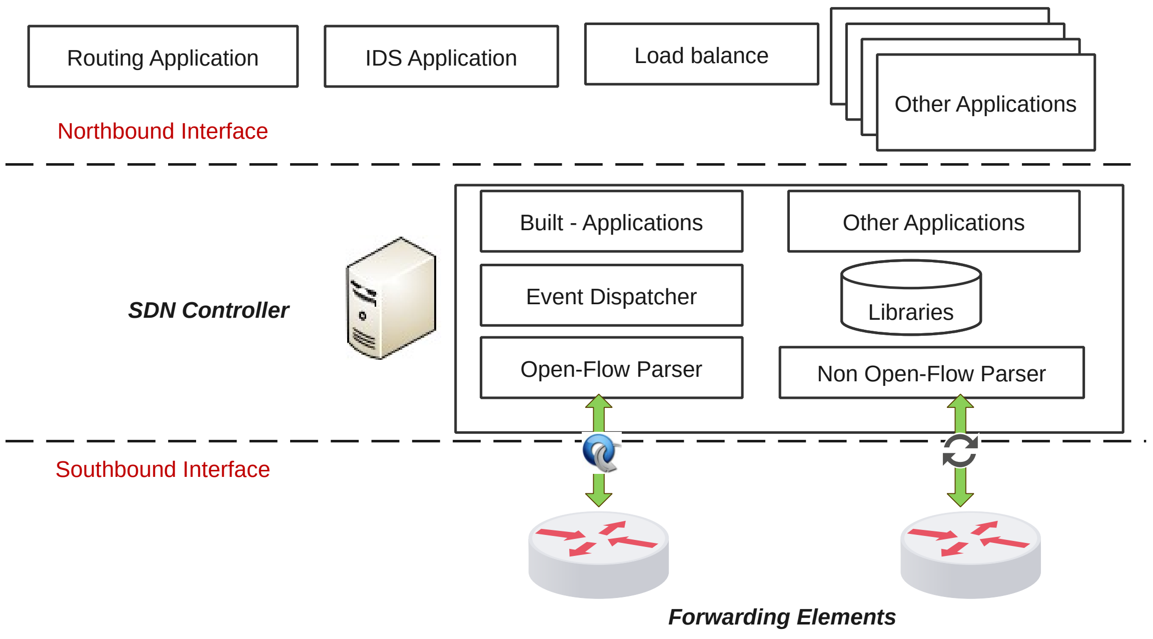


Figure 2.9. – The Basic Controller Architecture

module or inserting new modules. Thus, the SDN controller can perform analytics and orchestrate new rules throughout the network.

The SDN controller architecture enables many companies to form the SDN controller together. Each company can add its modules as a standalone unit that supports interoperability and flexibility. For example, Cisco provides a controller platform constructed by OpenDaylight. This open-source controller is interoperable with several different proprietary applications [IIAR20].

The SDN controller is a logical entity that receives instructions or requirements from the SDN application layer and applies them to the infrastructure layer's networking elements. Moreover, the SDN controller obtains network information from the hardware devices in the infrastructure layer and then transfers them back to the SDN applications with an abstract view of the network, including statistics and events. In addition to the developers' required applications, the basic SDN controller consists of four main components [AGS18]:

- **Built-in applications** that act as the basic applications for any controller to work, such as topology discovery or routing algorithms.
- **Parser** is responsible for recognizing the OpenFlow version or non Open Flow protocols such as XFlow (Netflow and Sflow), OF-Config, NETCONF.
- **Event Dispatcher** works to receive any incoming event as a packet and respond with creating a proper packet.
- **Libraries** support packet processing operations.

2.5 Discussion

As this chapter describes, the SDN has three layers. Each layer performs different duties, but all layer's duties are compatible. Firstly, the infrastructure layer is responsible for transferring the data between the connection parts. Secondly, the control layer manages the forwarding devices according to the predefined algorithms. Finally, the application layer draws the high policies of the network based on the administration's willingness. However, the application layer does not have a definitive standard yet, although there is some initiative to construct it. Therefore, the seen efforts mainly concentrate on developing the control and infrastructure layers.

In addition to what this chapter discussed, two other definitions should be mentioned, Although the work does not rely on them directly. But, understanding them provides a comprehensive view of the SDN principle. Firstly, SDN orchestration is a type of networking technology that enables automated processes in a network. It coordinates all the hardware and software components needed to support a given application or service. Secondly, Network Functions Virtualization (NFV) is a new structure concept that allows an organization to divide a physical network into various virtual networks. Both SDN and NFV concentrate on network abstraction. SDN separates the control plane from the Infurstracture plane. At the same time, NFV separates network forwarding and other functions from the hardware employing them. When SDN executes on an NFV infrastructure, SDN routes data packets from one network node to another. Moreover, SDN's control functions for routing, policy definition, and applications run in a virtual machine somewhere on the network. Thus, NFV provides essential networking functions while SDN manages and orchestrates them for particular applications [GTV$^+$14].

Chapter 3

Security Threats in Software-Defined Networking

According to chapter 2, the SDN has superiority over traditional networks in terms of network management because of the separation between the control and infrastructure layers. Despite all SDN advantages, the SDN is susceptible to new security threats and inherited types from legacy networks. This way, the attackers could exploit the SDN architecture to perform different malicious tasks.

In recent years, multiple studies about SDN security have reported the new security challenges and the SDN vulnerabilities that appear in the SDN architecture. Some researchers categorize the SDN vulnerabilities according to the fundamental functional layers as the application layer, control layer, and infrastructure layer [SB20]. Other researchers spoke about the potential type of attacks that the attackers can execute in these three layers of SDN architecture [EDP21] [ELKJ20]. However, the centralized control is still one of the most critical problems since it causes a single point of failure in the SDN architecture, especially when the control layer does not have a technique to sense the abnormal behavior.

This chapter describes the security threats that counter the SDN layers. It also covers the common dreadful attacks in the SDN, organizes state-of-the-art solutions addressing the DDoS attacks, and discusses the countermeasures.

3.1 Threat Vulnerabilities of SDN Paradigm

In the SDN, the idea of centralized control facilitates the programmable network process. It creates new vulnerabilities that provide different points to launch various types of security threats [BEFEE16]. Furthermore, all SDN layers have vulnerable aspects susceptible to different types of attacks. The SDN might counter the traditional methods that attack the legacy network elements such as the applications or infrastructure devices.

The side-effect of the traditional attacks could be more dangerous in the SDN than the legacy network, where these attacks are common and have a mild or moderate side-effect on the legacy networks. For example, once the malicious user promiscuously accesses a vulnerable machine or application in the legacy network, the side-effect would mostly be on the machine or a particular port. Therefore, the malicious user should maximize the privilege or exploits the victim machine to launch another attack against different machines or networks [DSR17]. In addition, other attack types exploit the new SDN architecture and work under the SDN environment only due to the separation between the data and control layer functionality. The malicious user directs the attack towards the SDN controller or the Southbound channels, which destroys the whole network. In general, four attack dimensions conclude the significant attacks against the SDN network as Figure 3.1 depicts.

1. **Attacks on the infrastructure layer elements**: The infrastructure layer consists entirely of connected dumb devices (OpenFlow switches) responsible for transferring information from the source to the destination. Thereby, the attacker strives to obtain unauthorized access to a machine or application in the SDN network to launch the attack. For example, the attacker sends many malicious traffics to exhaust the goal resources such as memory or CPU [ELKJ20].

   When the first packet arrives at the OpenFlow switch, the switch looks up the flow table to match the packet header information. In case there is no flow rule to match, the OpenFlow switch uses the `Miss Table` flow to forward the packet to the SDN controller as a *Packet_In*. Eventually, the controller installs a new flow rule in the flow table. Due to the space constraint, the limited resources of the switch become a problem; for instance, the OpenFlow switch memory is limited. Therefore, the attacker exploits this scenario to destroy the OpenFlow switch performance by installing more false flow rules.

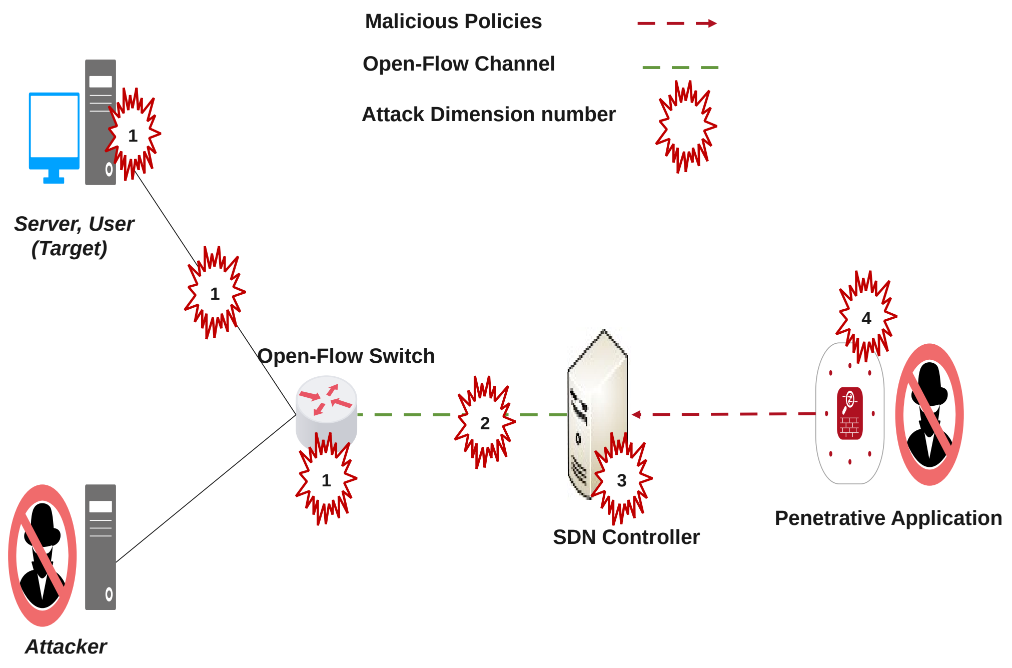


Figure 3.1. – SDN Attack Dimensions

Furthermore, the detection of false flow rules is a significant security challenge for the infrastructure layer. But, the OpenFlow switches are only packet forwarding devices, which means the OpenFlow switches do not have a process to distinguish between genuine and malicious flow entries. So, the OpenFlow switches depend on the SDN controller to ensure the security of the infrastructure layer. Hence, if the network controller is compromised, all infrastructure layer elements would be influenced and compromised [AS19b] [SB20].

2. **Attacks on the Southbound interface**: In the SDN network, The OpenFlow channel is the primary and only artery that the SDN controller uses to configure the devices of the infrastructure layer and supply the flow rules. Although all OpenFlow switches physically share the same channel, even though they logically seem to have a separate channel to the SDN controller.

   The attacker can generate malicious traffic, which forces the switch to launch a massive volume of *Packet_In* to the controller. Thus, a flooding attack causes congestion in the OpenFlow channel links and may expend high-data bandwidth [DAJ19]. That means the attack destroys the SDN network entirely because the

OpenFlow channel breaks down. Furthermore, the communication between the SDN controller and the switches would be unavailable. Finally, the SDN controller trusts information that comes from the switches through the OpenFlow channel. The attacker exploits the controller's trust to launch other attack types, such as sniffing valuable information or obtaining full unauthenticated access to the control layer [LBJ$^+$17] [AS19a].

3. **Attacks on the control layer**: The controller acts as the the brain of the SDN network and is mainly responsible for managing the network traffic in the infrastructure layer. Generally, the control layer is the center of the SDN network. It might be the most dangerous point in SDN security because it is a single point of failure. The adversary usually uses three methods to attack the control layer: (1) exploiting the Northbound interface, (2) exploiting the Southbound interface, or (3) using specific applications to control all infrastructure layer activates [DAJ19].

   The SDN network would be disrupted as long as the adversary made the SDN controller down or has unauthenticated access. Therefore, the attacker will try to focus all available malicious methods on the SDN controller. Significantly, the SDN controller has the same vulnerabilities as the installed operating system on it. Once the adversary successfully manages the SDN controller, the attack exploits the controller to forward traffics based on the rules that adversary sets up. In addition, the adversary can put his policy when the attack successfully exploits the vulnerable Northbound interface [ELKJ20].

4. **Attacks on the application layer**: The SDN applications implement various network-related functionalities, such as applying security methods and routing policies, performing network management, or running services. The application layer is located on the top of SDN network architecture. It is responsible for drawing the network's policies by interacting with the SDN controller through the Southbound interface.

   In addition, the application layer monitors the performance of the control layer and could decide to change the controller roles. As well, the control layer reports the infrastructure layer situation to the application layer to produce a high decision related to the network by the SDN applications. Once the intruder illegally accesses one of the SDN applications, all activities managed by the penetrative application would violate the network policy without any reaction from the SDN network.

   Moreover, the penetrative application could influence another application that is not the main objective of the attack and cause genuine security breaches in

SDN [ELKJ20] [DAJ19]. The attacker follows two approaches to attack the application layer. Firstly, the attack concentrates on specific applications to exhaust its resources. Secondly, the attack concentrates on the SDN Northbound interface to stop the communication between the SDN controller and the application layer [LBJ+17] [LSL+20].

Generally, the adversary aims to control the system by obtaining illegal access to the SDN resources. After that, the adversary tries to gain important information or destroy the network operations. The adversary explores the network machines and applications to start the attack, understands their activities, and gathers information about the aimed subject, such as the addresses (IP, MAC), its applications, and the used operating system. After finishing the reconnaissance, the adversary exploits the obtained information to scan the weak points (vulnerabilities). Thereby, the adversary attempts diverse attack scenarios against the target object. Once the adversary discovers the existing vulnerabilities, the illegal access is done.

When the adversary controls the victim machine using a remote shell (e.g. Trojans, Backdoors, Rootkits, etc.) to keep the victim under control as long as possible. Also, the adversary obtains more critical information or launches another attack against a different target in the network. Finally, the adversary should cover all signatures guiding the administrator or the security system to detect the attacker's identity. For instance, the attacker removes all the system logs or at least the relevant logs [ELKJ20].

3.2 Distributed Denial of Service Attacks

The DDoS attack is a spatial type of DoS attack that overwhelms the target machine by malicious traffics. Therefore, the attacker uses multiple connected devices, known as a botnet, to increase the traffic volume as much as possible. In contrast to other cyberattack types, the DDoS attack does not need to breach or control the target. But, the DDoS attack exploits the fact that the target has limited resources. Thus, the attacker aims to exceed the resources' capacity by sending a tremendous volume of malicious requests.

Consequently, the target will be unavailable to legitimate users or functioning incorrectly. Moreover, the attacker makes the DDoS attacks in short, long, continuous, or repeated bursts. Likewise, the attack could impact the target for hours, days, weeks, and even months. As a result, the targeted organization suffers from long-term reputation damage, works to recover, and spends fortunes compensating [SLHG19].

3.2.1 Classifications of DDoS Attacks

Generally, there is no standard method to classify the DDoS attack types. But, each research tries to introduce its vision based on the used criteria. This subsection organizes the DDoS attack classifications into several groups based on the targeted Open Systems Interconnection (OSI) layers, the methods to launch the DDoS attack, the volume of generated traffic, and finally, based on the attack rate dynamics.

Firstly, the DDoS classification is presented according to *the target layer in OSI*, where the attacker aims to destroy the top layer, which is known as the application layer attack. Like this, the DDoS attack affects the functionality of the applications and provides a grave experience to legitimate users. Generally, the attacker uses different protocols to execute this attack type. For instance, Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS), with the help of the Create, Read, Update and Delete (CRUD) operations. Typically, a web browser (zombie) interacts with the target machine by transmitting HTTP requests. Moreover, the HTTP requests are a "GET" retrieving statistical content such as images or "POST" requesting the target machine to perform some processing such as looking up items in a database.

Therefore, the attack imposes the target machine to allocate resources that serve the attack. Consequently, legitimate users would be denied to get the service because of the shortage of resources. However, the attacker directly concentrates the DDoS attack on the links or resources of the network devices located on the network or transport layer. Furthermore, the attacker could use a malicious program installed on a device (zombie) to send vast amounts of traffic to the victim through the transport or the network layer. Also, the attacker considers the protocols as vulnerabilities such as ICMP, UDP, and TCP [SDR21].

Secondly, the attacker could directly launch the DDoS attack towards the victim or use innocent intermediate nodes that reflect malicious requests. This classification defines the DDoS attack as *direct and reflector-based DDoS attacks*. In the direct type, the adversary's zombies launch the DDoS attack to the target immediately. But, in the reflection or amplification type, the adversary depends on the machine used to launch the DDoS attacks. For example, the adversary uses the servers running UDP-based services to reflect a huge volume of DDoS attacks by crafting the source addresses to be looking like the victim.

Thereafter, the adversary sends malicious requests to the server that replies to the victim by sending messages which its volume is too huge. For example, in Domain Name System (DNS) amplification attacks, the attacker sends many queries about some web server addresses. Still, the attacker spoofs the victim's IP addresses as the source address. Thus, the DNS server directs the replies to the victim. Another example, Network Time Protocol (NTP); the attacker also spoofs the victim's IP address as a source address and then sends many requests to the NTP server to adjust the victim's local time. Finally, the server will replay the response to the victim [BSS17].

In some cases, the DDoS attack could be classified into *Direct and Indirect DDoS Attacks*. The adversary does not target the victim directly but attacks the links or other essential services that support the victim's functionalities to remain working. This indirect attack degrades the victim machine performance at the highly paramount services of the victim. For example, crossfire (Many attackers flood a specific link with a massive volume of false requests) [ASAR16]. Another example is coremelt (To flood a specific link, the attackers only send traffic between each other) [uRWA$^+$20] attacks.

In addition to the classifications as mentioned earlier, the DDoS attack might be classified based on the used traffic volume into *High-Rate and Low-Rate DDoS Attacks*. Typically, the adversary submits extremely malicious traffic (high-rate DDoS attacks) to flood the victim. Sometimes, not all high-rate traffics is recognized as attacks because it could be a flash crowd. However, the attacker can send malicious traffic at a low rate to match the legitimate traffic behavior. This malicious traffic requests the victim resource to perform intensive processes (e.g., CPU-intensive queries) [AT21].

Ultimately, the traffic characteristics could be considered to classify the DDoS attack, such as *the dynamics of the attack traffic rate*. Generally, the malicious traffic characteristics exploited in the DDoS attacks have four categories (Constant-rate, increasing, pulsing, and sub-group attacks).

In the constant rate attack, the attacker launches malicious traffic and maximizes the rate in a short time. Besides, the used zombies launch the attack after receiving a command from an attacker, which makes a sudden packet flood at the victim resources. Hence, the DDoS attack is easily detected. Therefore, the attacker changes the used strategy to gradually increase the malicious rate, which gives the attacker more time to understand the victim's response.

In a pulsing attack, the attacker uses bots intermittently instructed to launch malicious traffic towards the victim. So, the pulsing attack prevents the attacker from being detected. In the last category, the sub-group controls several zombie sets to attack the same victim. The attacking groups are activated and deactivated arbitrarily. Therefore, the attacker stays disguised and attacks for a long time [Bha19].

3.2.2 SDN as a Victim of DDoS Attacks

Although the SDN concept brings many benefits (see subsection 2.1.5), the SDN architecture introduces compromised aspects to different security issues. For instance, unauthorized access, data leakage, flow rule modification, malicious application implementation, etc [DAJ19].

Recently, the DDoS attack is the most challenging threat that faces the SDN because the adversary exploits the separation between the control and infrastructure layers to target the SDN components. Moreover, the new architecture contains a single point of failure in some vulnerable points that spoils the SDN architecture and is a potential point for the attackers [PPS19]. This segment demonstrates five DDoS attack types that the intruder launches by exploiting either the SDN paradigm or the inherited attributes from the traditional networks as the following:

1. **Control Layer Saturation**: The SDN controller machine has limited resources and acts as a single point of failure. So, the attacker directs the DDoS attack towards the SDN controller resources such as the memory and CPU. To launch this type of DDoS attack (which is known as saturation attack, *Packet_In* Injection attack or *Packet_In* flooding attack) in the SDN, the attacker exploits the `Miss Table` to force the OpenFlow switch to send malicious *Packet_In* messages towards the SDN controller. Once the attacker crafts the packet header to be convenient for the `Miss_Table` entry, the attacker floods the crafted packets towards the switches.

   Thereafter, the switch uses the `Miss Table` to forward all the received packets as *Packet_In* to the centralized controller. As a consequence, the controller would be busy to satisfy the false *Packet_In* request and prepare a forwarding rule for the new flows. Thereby, the false *Packet_In* exhausts the centralized controller resources and processing power. In the end, the saturation attack overloads the SDN controller and makes the SDN controller unreachable to legitimate users because the SDN controller works only on the false *Packet_In* requests. Resultantly, the Saturation Attack downgrades the whole SDN architecture [XWX20] [DGLG17] [SB20]

2. **Congestion of Southbound APIs**: Once the attacker launches the saturation attack toward the controller, the switch forwards bulk *Packet_In* messages to the controller because there is no entry matching that crafted packets. In fact, the OpenFlow switch sends some part of the packet as *Packet_In* message, and the remaining parts are buffered in the switch.

If the buffer space is insufficient to keep more packets, the switch transmits the entire packet through the Southbound API. That leads to creating a curtailment in the channel and hence making the channel a bottleneck for the legitimate *Packet_In* messages [Opea] [AS19b].

3. **Buffer Saturation**: The attacker targets the OpenFlow switch buffers, in addition to the SDN controller resources and the Southbound channel. Each switch's port has two queue buffers, one for incoming packets and the second for outgoing packets. The switch uses them to wait for the busy resource such as the switch's CPU. The adversary aims to make the queues buffer full by flooding the switch with a rapid and massive volume of packets. Furthermore, when the malicious packets target the SDN controller, the OpenFlow switch sends the complete packet as a request to the controller as long as the memory buffer is full.

   The SDN controller receives and buffers the *Packet_In* messages through port of the OpenFlow channel until the routing algorithm becomes available to process them as *Packet_Out* or *Flow-Mod* messages. But, the *Packet_In* messages are automatically removed from the buffer after some time (expired time) if the routing algorithm is busy [SB20] [Opea].

4. **Table-Miss Striking or Flow Table Overflow**: In the SDN environment, the standard method to forward the packet towards a suitable port is realized by matching the packet with the flow rules of the flow table. But, when the OpenFlow switch does not find the proper entry, the switch requests a new flow rule from the SDN controller. Hence the new flow rule is stored in the flow table. The flow rule reserves a part of the memory space for a fixed time before the OpenFlow switch removes the flow rule [AS19b]. The OpenFlow switch uses Ternary Content-Addressable Memory (TCAM) to store the flow rules given by the SDN controller in the flow table. The TCAM has a limited capacity to store the flow table entries, in addition to its high cost and power consumption [PDFN17] [AS19b].

   The adversary exploits the TCAM drawbacks by overwhelming the flow table with false flow rules. It either floods the controller with *Packet_In* or launches a slow DDoS attack to overtake the controller's defense mechanism [SPT$^+$18]. Consequently, the attack would quickly run out of the flow table memory, and the false flow rules fill the flow table. Also, the attack removes the normal entries, and hence the OpenFlow switch's performance downgrades until being out of the service.

5. **The Inherited DDoS attacks from the Legacy Network**.

   In addition to the aforementioned DDoS attacks, the adversary can use the infrastructure layer to target the traditional DDoS objectives such as hosts, servers, applications, routers, etc [Bha19]. Throughout the history of Communication networks, the adversary always develops different approaches to inject the network with crafted packets. As long as the adversary can scan the necessary information and then use it to craft the packet, the traditional network objectives will stay under the threat of the DDoS attack.

   Moreover, the attacker might be selfish, surreptitiously piggybacked on authenticated user's resources to reduce costs and even obtain free services known as a Freeloading attack. This way, the attacker might maliciously damage the target by launching DoS or DDoS attacks on the remote target. To set up the freeloading attack, the attacker eavesdrops on the transmitted packets using the promiscuous mode to obtain sufficient information about the traffic flows, users, and route entries in the network. Thereafter, the attacker sends malicious packets with crafted addresses pretending to be a legitimate user. As a result, the attacker exploits the available resources illegally and may exhaust available resources and services in the infrastructure layer [CLMR19] [PCK16].

3.3 Benefits of Using SDN Against DDoS Attacks

In the traditional network, the DDoS attack grows and increases in size, frequency, severity, and also become more sophisticated. The available IDSs are limited to detect DDoS attacks or alleviate their side effects. While the attackers constantly refined new approaches to exceed the existing IDSs [BS16]. Even though the SDN architecture is compromised to the DDoS attack, as mentioned in the previous subsection, the SDN has distinct features assisting in defeating DDoS attacks. Therefore, the SDN platform shows superiority over traditional network paradigms in detecting and mitigating DDoS attacks.

The separation between the control and the infrastructure layer is a perfect solution to large-scale experiments that are difficult and complex in legacy networks. That means the SDN concept facilitates the investigation of extensive mechanisms since the dynamic configuration promotes the experiment environments. For instance, once the developers need to examine a new algorithm, they update every forwarding node individually because the control logic is embedded in the forwarding node of the legacy network. Consequently, the advanced initiative is easily deployed and transferred from the trial phase to reality [WGH$^+$14]. Furthermore, the SDN controller contains written intelligent scripts (e.g., IDSs) that direct the infrastructure layer traffic and obtain information about the ongoing traffics. In addition, written scripts can process and analyze the traffic to define the malicious traffic and its source; thus, the SDN controller maintains the network's performance.

Network intelligence is logically centralized and located in the SDN controller that maintains a global network view. Therefore, the controller can analyze and monitor the network's traffic. In other words, the controller smoothly installs the appropriate security policies preventing threats based on the global view of the network. Moreover, the SDN controller distinguishes between the compromised and benign hosts by the given information about hosts [UJ20].

Besides, the SDN controller or the application layer contains various application utilities that can analyze the network traffic based on different software tools, modules, algorithms, and databases. So, the traffic analysis-based software allows the innovation and minimizing the switch's load to parse the traffic.

The SDN architecture is open and programmable because the application layer draws the network policies by programming the algorithms in the SDN controller. The programmability of the SDN network supports the control layer to quickly responds to abnormal events or sudden problem and readily resolves the problem at any point of the network [CIV20].

The SDN network dynamically updates the policy based on the traffic analysis. Therefore, when the SDN controller responds to abnormal traffic behavior such as the DDoS attack, it installs a new flow rule or modified the pre-installed rule. Also, the SDN concept instantly facilitates propagating new innovative algorithms across the network to detect and mitigate malicious traffic. On the contrary, the legacy network applies the mitigating flow rule on the target host only. Likewise, the network engineers manually configure the legacy switches to change a particular network policy [UJ18].

In dependence on the aforementioned discussion, Table 3.1 briefly summarizes why the SDN architecture is suitable for combating DDoS attacks or even other security threats.

The reasons	**Benefits**
Separation of the control layer and infrastructure layer	Operating the new algorithms and innovation on the large-scale experiments easily
Global view of network	Supports the traffic monitor and prevent the security threats
Traffic analysis based on software	Using the written application to control and analyze the network behavior
Network Programmability	Rapid response to abnormal events or sudden problems
Dynamic network policy update	Dynamically and instantly updates the policy based on the traffic analysis

Table 3.1. – The SDN Reasons to Effectively combat the DDoS Attack

3.4 Solutions to Address the DDoS Attack in SDN

Recently, the detection and mitigation of the DDoS attack in the SDN environment have been an active area of research, and various approaches are carried out as potential solutions for the DDoS attacks. The related work could be categorized into several categories according to the detection metric and the used mechanism.

This section discusses the four classifications for the suggestions that address the DDoS attack in SDN. In the first category, the intrinsic suggestions concentrate on the SDN components and their functional elements. Secondly, the statistical solutions depend on information theory-based DDoS defense solutions. The third and fourth categories are introduced using Machine Learning or Artificial Neural Network(ANN)-based defense solutions that concentrate on the network traffic and the flow characteristics.

3.4.1 Architecture-based Solutions

Generally, the researchers often work to decouple some of the SDN controller functions to reach better performance. For example, decoupling the monitoring and controlling utilities of the SDN controller will extend the SDN architecture by adding new devices. FloodGuard, in [WXG15], is a clear example of the SDN controller decoupling where it adds two modules to the SDN architecture: a proactive flow rule analyzer and a packet migrator. Both modules would be silent until the SDN controller detects a DoS attack situation. Hence, the proactive flow rule analyzer creates and installs flow rules proactively to the affected OpenFlow switches. Besides, the migration module simultaneously directs the `Table Miss` packets to the infrastructure layer's cache device. As a result, the work decouples two essential parts, the controlling and monitoring, to obtain a better reaction in the DoS attack conditions. In fact, this solution has no mechanism for recognizing the DoS/DDoS attack, but the work's design provides more stability and resistance to operate when the DoS attack is detected. Also, the work could introduce delay to the network response.

In [JASD12] Jafarian presents the OpenFlow random host mutation method, where the SDN controller assigns random and dynamic IP addresses to the hosts under its domain. The SDN controller protects the hosts from external and internal attacks and scans. To do that, the SDN controller connects to a Dynamic Host Configuration Protocol (DHCP) server that provides the requested IP addresses. After that, the translation to actual IP addresses is executed. The solution effectively covers the host's location due to continuously changing the IP addresses, making forwarding the DDoS attack toward the target partially hard. But, it would be hard to manage the situation when multiple attackers request multiple hosts simultaneously, particularly how the DHCP server can react and respond.

Avant-Guard, in [SYPG13], protects the TCP servers in the infrastructure layer against the SYN flooding attack inspired by the SYN proxy, which handles TCP connections in a middlebox. The Avant-Guard adds a new device to the infrastructure layer to be intermediate between the OpenFlow switch and the controller. The Avant-Guard has two modules, the connection migration to filter the malicious SYN request before forwarding any notification to the control layer. Suppose the switch receives a TCP SYN packet from a new address, and the packet does not match any flow rule in the flow table. In that case, the infrastructure layer answers with a TCP SYN/ACK packet to the sender. Once the sender submits a TCP ACK packet to complete the TCP handshaking process session, the controller is notified and completes the process. The second module is the actuating trigger, enabling the switches to asynchronously report network status and payload information to the SDN controller. Also, actuating triggers activates a predefined flow rule under some conditions; hence it supports the SDN control to manage network traffics without delay. Although the solution prevents the suspicious hosts from accessing the network, it does not handle the other transmission protocols such as ICMP or UDP. Moreover, the use of connection migration adds more delay to the network connections.

Wang et al. [WC17] introduce SGuard as a security application on top of the SDN controller. SGuard includes three modules: (1) Access control module that tracks the identity of the connected users, collects user's information (such as MAC, IP, Port, and Switch ID), and binds the collected information in Hashtable. (2) Classification module that requests flow entries from flow tables and extracts certain features immediately. Each sample is classified either as normal or attack traffic based on Self-Organizing Maps (SOM) (SOM is explained in Appendix A). (3) Infrastructure layer cash which is located between the infrastructure layer and the SDN controller. It temporarily caches the `Miss Table` packets classified as malicious traffic during the saturation attack from the switch. However, the cost of the memory is expensive, and the SGuard could be a bottleneck under the `Miss Table` saturation attack.

Chen et al. [CJS+16] propose SDNShield, a device connected to certain OpenFlow switch to protect both the edge switches and the SDN controller from DDoS attacks. Once the *Packet_In* messages exceed 80% of the link rate, the controller transfers the `Miss Table` packets to the SDNShield. The SDNShield has two units; the first unit is the attack mitigation unit which executes two stages. The first stage, statistical differentiation, identifies legitimate flows with low complexity by building profiles for the normal predictive behavior compared to the suspicious behavior by a conditional legitimate probability and a Bayesian-theoretic metric (statistical approach derives probability distributions for sets of variables). Finally, the first stage decides to reject the flow or forward it to the SDN controller. The second stage aims to decrease the false positives of the rejected flows by TCP connection verification to ensure that all legitimate flows are accepted. Nonetheless, the solution proposed is to connect attack mitigation units to every edge switch. Also, the second stage does not include the other transmission protocols such as UDP and ICMP.

Manso and others also suggest adding IDS as standalone units in [MMS19]. The work relies on three essential architecture components, the infrastructure layer, the SDN controller, and the IDS unit (called snort). The IDS receives a copy from all packets reaching the monitored port. Then, it uses the snort's rules to analyze the exchanged traffic. Eventually, it ensures that the traffic does not exceed the statistical threshold. When the IDS detects malign traffic, it sends an alert to the SDN controller, which installs the new rule to block the source of the DDoS attack. In general, mirror all packets is helpful with small or home networks. Still, it is not successful with the extensive network because the snort unit will be a bottleneck point, primarily when the administrator monitors many ingress ports in the network.

Safe-Guard Scheme, in [WHT$^+$19], are suggested to protect the control layer against the DDoS attacks. When the OpenFlow switch submits abnormal traffic to the affected controller, the attack mitigation scheme transfers the traffic to another SDN controller. Multiple SDN controllers reduce the burden on the attacked controller and make the control layer more scaleable. The responsible controller detects anomaly behavior using the switches' byte rate and variable rate of asymmetric flows. The other controllers successfully identify the affected controller based on the anomaly detection alert message about a specific switch. But this technique would fail if one of the controllers is hacked and making false rules in the switches. Then, the switches would not recognize the malicious behavior.

3.4.2 Statistics-based Solutions

Analyzing and evaluating the imported information from the network traffics is the fundamental material for the statistics-based approaches. In the SDN environment, the objective of this type is to increase the immunity of the SDN controller against malign attacks by analyzing and collecting statistics related to traffic flows. Commonly, the researchers use either (1) information theory-based entropy (E) according to equation 3.1, where a variable x defines a specific feature of a new incoming packet and P is the appearance's probability of x (subsection 4.3.2 presents the entropy in details), or (2) mathematical equations (e.g., rate average, fixed boundaries, load percentage, etc).

$$E = -\sum_{i=1}^{n} P(x_i) log(P(x_i)) \tag{3.1}$$

The entropy shows the randomness level in the network features and represents uncertainty measurement. Moreover, the entropy discovers the difference's value between the current and expected network conduct. The researchers introduce the similarity of two probability distributions as the divergence metrics (a statistical equation that uses two probability distributions as input and returns a number. This number measures to what extent they are different. The number must be non-negative and equal to zero if and only if the two distributions are identical. Bigger numbers indicate a greater dissimilarity. Many equations represent the information distance among the traffic flows to capture the abnormal traffic behavior).

In fact, using statistics-based approaches to handle the DoS/DDoS attacks is efficient because it provides a lightweight solution to defines the network conditions. Furthermore, the statistics-based approaches obtain accumulative information about the incoming requests and level of the resources consumption. As well, it has very little computing overhead and can handle a large number of packets. However, the statistics-based approaches cannot always guarantee to detect the DDoS attack. Still, they are adequate for assessing network traffic behavior. Additionally, the entropy does not consider various factors when calculating the probability distribution of a specific traffic feature [EDP21].

In [DDZX16], the work suggests collecting information from the incoming flow in the edge switches. Thereafter, the work calculates the number of packets that use the same flow rule to classify flows as low-traffic or normal-traffic flows according to the threshold. The low-traffic flow is what had few data packets and needs high consumption of the SDN controller resources. Consequently, the low-traffic flow could lead to making the SDN controller unreachable. The work applies the Sequential Probability Ratio Test (SPRT) to minimizes the false-positive rates. SPRT better defines the ingress ports connected to a large number of low-traffic flows. (SPRT is a hypothesis test for Sequential sampling that works in a very non-traditional method; instead of fixed sample size, SPRT chooses one item or more every time and then test the hypothesis).

PacketChecker module is suggested in [DGLG17], where the controller stores the users' information in a mapping table that includes the switch port and user's addresses. Once the SDN controller receives a new *Packet_In* message that has part of its information in the mapping table, the SDN controller drops the message. But, if all the information is new, the controller accordingly adds a new row in the mapping table. Hence, this work alleviates the consumption of the SDN controller resources. Nevertheless, the attacker can use this approach to saturate the SDN controller memory by targeting the mapping table. Alternatively, the attacker could overtake the SDN controller by spoofing the complete information requested of the authenticated user and then launching the DDoS attack on the infrastructure layer elements.

Giotis et al. [GAA$^+$14] depend on extracting the features from the packets reached to a specific host (e.g., Server) to analyze the randomness level using the entropy method. sFlow device collects the traffic of the monitored host as a unit embedded in the SDN controller. The SDN controller calculates the entropy of the ports and IP addresses for both destination and source to detect the anomalies. Then, the anomaly detection module inspects all flow entries for every time window and identifies the malicious flows to detect the anomalies. In the end, the controller installs a flow rule to block the attacker.

Swami et al. [SDR19] consider entropy as the main value to deal with *Packet_In* under malicious traffic. Furthermore, the window size to calculate the entropy is equal to a certain number of *Packet_In* messages and their respective destination IP addresses. Xu et al. [XWX20] propose SDNGuardian. The author defines the sensitive fields of the *Packet_In* messages that influence the performance of controller applications. So, the statistical analysis approach verifies which fields are essential to the applications to generate flow rules. Hence, the entropy of the sensitive fields is calculated to determine if a DDoS attack occurs based on the entropy result. Thereby, the work determines the attacked ports based on the information gain (which port has the biggest influence on the entropy value) and limits the rate of these ports. In the last step, a frequency-based filter is applied to the packets coming through the attacked ports. Hence, the SDN controller would delete the malicious flow rules based on the asymmetric characteristics of attack traffic.

Cui et al. [CWLZ19] introduce a work for detection of the DDoS attack using cognitive-inspired computing with dual address entropy. Two modules are used; the Statistic collection module gathers and calculates the frequency of each source and destination IP. Also, the feature computing module obtains two entropy values according to destination and source addresses. The DDoS attack is confirmed if the destination entropy is lower than a threshold and the source entropy value is higher than the normal traffic threshold. Then the SDN controller removes all relevant table entries to mitigate the DDoS attack.

Wang et al. [WJJ15] implement the entropy theory to detect the DDoS attack. The work depends on obtaining the traffic statistics and analyzes the network traffic arriving at the network devices. As IDS, the proposal locates the used algorithm in the edge switch connected to the hosts directly. The work calculates the entropy-based on IP and switches identity; to announce the DDoS attack when the entropy is less than a predefined threshold. Nevertheless, this solution needs to modify a switch which is not feasible.

Joint Entropy-based DDoS Defense Scheme in SDN (JESS) is described in [KAGA18] as a new method to protect SDN networks using joint entropy. The research suggestion includes three phases, (1) nominal, (2) preparatory, and (3) active mitigation stage. The nominal is supposed to be the attack-free phase to calculate the baseline information. Afterward, the SDN controller generates the nominal pair profiles for all traffic converted to the SDN controller. To prepare the profiles, the controller uses the obtained information from the transmitted packet by the hosts. The second phase detects the congestion if the bandwidth rate of the link between the one host and the connected switch is higher than a threshold. So, the SDN controller would activate the preparatory phase. Then, the OpenFlow switch starts transmitting packet header information to the SDN controller again. The SDN controller initiates a new pair profile and then calculates the joint entropy of pair profiles to compare the results with the predefined value in the nominal phase. Once the SDN finds the difference between the entropy's exceeds that threshold, the DDoS attack is confirmed. In the last phase, the controller installs flow rules to drop the attack packets. In fact, this method would increase more burden on the SDN controller, particularly when the attacker launches DDoS from multiple sources using bots.

Jiang et al. [JZZC16] present Entropy-based DDoS Defence Mechanism (EDDM) to handle DDoS attacks. The EDDM is an extended SDN controller. The solution is divided into three stages: window construction, DDoS detection, and DDoS mitigation. The method calculates the entropy metric to differentiate the network traffic into normal or malicious traffic. Then, the solution finds the relevant edge switch Inport to block the attacker. Still, the entropy is not sufficient to differentiate between the flash crowd and the DDoS attacks.

3.4.3 Machine Learning-based Solutions

Machine Learning algorithm (MLA) aims to make the software application predicates the outcome based on learning from the previous discrete values. With the MLA, the machine obtains an experience from the historical information patterns to distinguish between normal and abnormal behavior. Therefore, MLAs have been applied in the SDN security to detect DDoS attacks. MLAs have a high accuracy in recognizing the attributes of the traffic conduct better than statistics-based detection solutions [BS19]. Typically, MLA must be first trained on attack-free flows to detect anomalies in traffic more accurately. It is noteworthy that Appendix A has a short description of the algorithms mentioned in this subsection to make it more readable.

In [PP18], the K-Nearest Neighbor (KNN) algorithm has been implemented. The used features are (number of hosts connected to the requested host, number of packets, delay in millisecond, protocol, throughput, source IP, destination IP). The system model collects information from the switches every 10 seconds to check whether the attack is present or not. Thereafter, the KNN algorithm learns how to determine the malign and benign behavior to be ready as an IDS unit in the controller to detect and classify anomaly traffic.

Other authors present solutions based on hybrid MLAs, for example, in [PBP16]. The normal and abnormal flow behavior information is extracted from the OpenFlow table to check whether there is an attack or not. Thereby, the IDS unit in the SDN controller is responsible for collecting information from the flow tables periodically and extracting the requested features. The Support Vector Machine (SVM) and a self-organizing map (SOM) are combined to detect and mitigate attacks in two phases. Firstly, SVM classifies flow rules into a malign or benign flow. But, when the flow's position is located between two margin lines or a vague region in the linear SVM representation (which means that the decision could be false positive). Therefore, the same features are forwarded to SOM to make a final decision. In the end, the DDoS attack classifier and a policy enforcement module are executed on the malign flows to mitigate the attack.

In [DSB21], the author presents a method to detect DDoS attacks in an SDN through MLA and a statistical method. The method comprises three sections: collector, entropy-based, and classification sections based on MLA algorithms (BayesNet, J48, Random Tree, logistic regression, REPTree classification algorithms). The work achieves good results in terms of the accuracy in detecting DDoS attacks in SDN, where several datasets are applied: the UNB-ISCX, CTU-13, and ISOT datasets.

Prasath et al. [PP19] proposed an agent program framework that protects the OpenFlow switches against external attacks. Furthermore, they used a meta-heuristic bayesian network classification algorithm to classify the incoming packets into benign or malicious. Santos et al. [SSS$^+$20] implemented and analyzed four MLA (multiple layer perceptron, SVM, Decision Tree, and Random forest Algorithm(RFT)) to classify DDoS attacks in an SDN environment. They adopted the Scapy tool to produce traffic in their simulation. As an outcome, they claim that the Random Forest algorithm showed the best accuracy and the Decision Tree algorithm had the best processing time. Myint et al. [MOKKV19] propose a solution that applies advanced SVM to detect the DDoS attack with minimum overhead. Moreover, the proposal depends on volumetric and asymmetric features to minimize training and testing time. This approach successfully deals with two types of flooding attacks: UDP flood and ICMP flood.

To recognize the malicious traffics conduct, the graph model is implemented by Bing Wang et al. [WZLH15]. The algorithm has relational graphs for different traffic patterns that allow the model to determine whether the traffic is malicious or normal. The algorithm creates a relation graph for the new traffic and compares the graph to the stored graphs. In the end, the SDN controller installs flow rules to block the source IP of malicious traffic. However, the MLA models mainly exhaust time, device power, and processing resources when the model aims to make the correct decision about the malign traffics through grouping flows, clustering, and redirecting. In addition, machine learning-based solutions need to be trained against a wide range of traffic patterns. In the SDN environment, the management's challenge appears to organize the communications between the SDN controller and the located MLA, which surly imposes more delay in treating the benign requests [EDP21].

3.4.4 Deep Learning-based Solutions

Deep Neural Network Algorithm (DLA) is a subset of MLA. It uses a specific technique to improve a machine's ability to recognize and amplify minor patterns. The deep neural network algorithms mostly contain three-layer types to shape the hierarchical structure of the model as input, hidden, output layers. The following Figure 3.2 represents an N layered Deep Neural Network.

In the input layer, the model receives values of the unlabeled features. Each feature enters the model through a neural node that applies a mathematical equation (activation function) on the feature value. Then, the neural node submits the result as input for the neural nodes in the next layer (hidden layer). According to the developer adjustment, the input layer nodes could be connected to the subsequent hidden layer nodes either

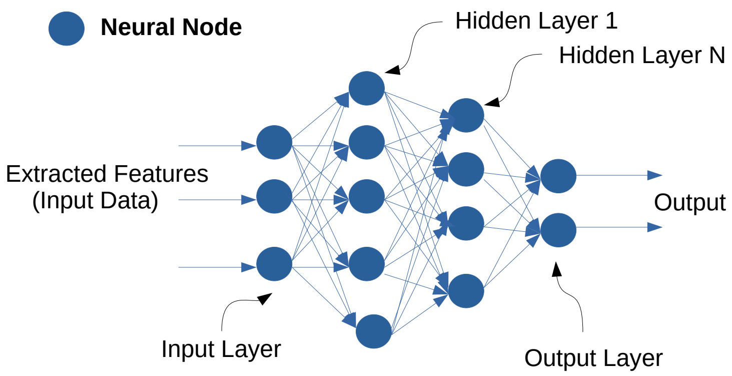


Figure 3.2. – Neural Network

fully or partially. In other words, the neural node might submit the result to all the subsequent hidden layer nodes or specific nodes. Once the hidden layer node receives the submitted values and applies its activation functions, it submits the new result to the next layer. Which is another hidden layer or the output layer provides the final prediction.

It is noteworthy that all neural nodes have a threshold to compare with the received value to pass the output or ignore when the value is greater than the threshold. Moreover, the neural node could receive many values from different nodes in the successive layer that acts as the input layer for the predecessor layer. Each value has an associated weight that defines the input's influence on the activation function. In DLA, the developers initially assign random weights for the received values. Still, the assigned weights would be automatically adjusted during the model training and even during the work, which is known as the back-propagation. Ultimately, the DLA enhances its performance and predicts a correct output based on the updated weights.

Although MLAs achieve a perfect improvement in the DDoS attacks detection, DLAs presents better treatment and solve the shortcoming aspects of the MLAs. In MLAs, human intervention is highly required compared to DLAs, which are more complex to launch and require minimal human intervention. Also, DLAs need more complex devices to work, such as the Graphics processing unit (GPU), which is wildly executed to optimize matrix operations efficiently. Conversely, the MLA can work in conventional devices like the Central processing unit (CPU). As a result, DLAs need more time and samples to train the model perfectly than the MLAs that can work with small samples.

However, DLAs work faster during the test and require very little time to predicate the result.

Moreover, the DLAs treat the data features at a high level, such as video, pixels, audio, etc., and do not need an external intervention to transfer the data to numerical values. Therefore, the accurate result of the MLAs depends on how the features are chosen and extracted. But, DLAs can enhance their performance with more samples, although the features' lack [XKL+18].

Moreover, DLAs also have more advantages paying the scientists and researchers attention to use against the DDoS attacks. For instance, the DLAs are self-learning, self-organization, better fault tolerance and robustness, and parallel performance. Besides, the DLAs identifies unknown attack patterns in addition to not only existing attack patterns. For the remaining of this subsection and to make it more readable, Appendix A has a short description of the algorithms mentioned.

SD-Anti DDoS mechanism is introduced et al. [CYL+16] to detect the DDoS attacks in SDN. The proposed framework contains four modules: (1) Attack Detection Trigger, which is responsible for verifying if the *Packet_In* messages number reaches a predefined threshold. Then, the module calculates the velocity Using exact-STORM, (2) Attack Detection works once the velocity result denotes an abnormal behavior. So, the SDN controller requests the flow table and starts extracting five parameters (number of packets per flow, number of bytes per flow, duration, packet rate per flow, byte rate per flow) to use them as input to Back-Propagation Neural Networks (BPNN) algorithm. After that, (3) Attack Trace-back defines the source of the DDoS attack, and (4) Attack Mitigation module blocks the attacker and deletes all relevant flow rules from the flow tables.

Li et al. [LWY+18] present Convolutional Neural Network (CNN), RNN, and Long Short-Term memory (LSTM) as detection models for the DDoS attack based on extracted features (source port, destination port, source IP, destination IP). The Detection models are trained using ISCX 2012 dataset and collect all packets received by the OpenFlow switch. Hence, the model decides whether the packets entered in the current OpenFlow switch are malignant packets or not. Consequently, the switch forwards the malignant packet to the Information Statistics module, which helps the Flow Table Generator module. Finally, the Flow Table Generator module determines the flow entries and priorities removed from the OpenFlow switch.

In [NPK+18], the SDN controller periodically requests the flow table from the switches and extracts six features (entropy of source IP, entropy of source port, entropy of destination port, entropy of packet protocol, total number of packets). The trained SOM and KNN algorithms use the six features to classify the network state as normal or under attack. In the attack presence, the used algorithm alerts the mitigation module. As a result, the mitigation module prepares and installs the combating flow rules to the switches.

Novaes et al. [NCLP20] use a hybrid scheme to detect DDoS and Port scan attacks in SDN networks. The suggested solution has three phases: (1) Characterization that extracts four features from the flow tables. It calculates entropy metrics to quantify the network attribute (source IP entropy, destination IP entropy, source port entropy, destination port entropy). In the detection phase (2), the LSTM model identifies the normal traffic signature. Also, fuzzy logic detects the anomaly in the network. The last module is the mitigation (3) creates the new flow rules for mitigating the DDoS attack.

3.5 Discussion

This chapter specifies two directions that the adversary follows to attack the SDN architecture. Firstly, the adversary targets the SDN controller because it is a single-point failure. Secondly, the traditional targets are inherited from the legacy network. All solutions concentrate on the first direction and try to counter it. But, few solutions attempt to face the second direction. Also, it is rare to find a solution that introduces IDS dealing with both together, according to our knowledge.

Recently, many works implement MLAs and DLAs as an IDS to detect DDoS attacks in the SDN environment. Each solution depends on several features to analyze the network traffic. Also, the proposed IDSs cover particular conditions according to the used dataset. The trade-off between the MLAs and DLAs still needs more work and enhancements, especially since their implementation in SDN is a partially new and hot research topic.

In both machine and deep learning, engineers use software tools that enable the SDN controller to identify the trends and characteristics of the network traffic by learning from an example dataset. In MLAs, the model uses the training data to train. Then, the controller can classify the test data, and ultimately real-world data. The development of features is essential by adding other metrics derived from the raw data, enabling the model to be more accurate. In DLAs, engineers and scientists skip the manual step of creating features. Instead, the data are fed into the DLA, which automatically learns the most valuable features to determine the output.

Chapter 4

Proposed Intrusion Detection System for SDN

This chapter describes the proposed IDS methodology to safeguard the SDN against DDoS attacks. Accordingly, the proposed IDS considers the new and inherited vulnerabilities in the SDN architecture that the adversary exploits to launch the DDoS attacks. Thus, The proposed solution handles the attacks that target the control and infrastructure layers' resources. Three new components are implemented to monitor the entire network behavior individually. Besides, the SDN controller obtains the components' feedback and makes the security decisions regarding the network status.

Section 4.1 introduces the proposed IDS framework to show the performance's integration among the used components. Afterward, section 4.2 clarifies the external device role, while section 4.3 explains the CNN component. Besides, section 4.4 describes the suggested method to utilize the IPSec concept. As well, section 4.5 presents the functions of the used RNN models to detect DDoS attacks.

4.1 The Proposed IDS Design Framework

Initially, the SDN controller can obtain information about the network conditions in three approaches. Firstly, the data is carried by the *Packet_In* messages (discussed in subsection 2.2.2). Each *Packet_In* transports information; such as the packet source (e.g., sender's ports, MAC and IP addresses), the used protocol, information about the switch that sent the *Packet_In*, the packet destination (e.g., destination's ports, MAC and IP addresses) and the match fields [Opea]. In the second approach, the SDN controller requests the flow table statistic from the switches (discussed in subsection 2.2.2) such as the flow tables, meter table, group table, and ports statistics (packet numbers or the data amount). Thirdly, the administrator could depend on external devices to collect information and report them to the SDN controller about the network status (as mentioned in 3.4.1). Consequently, the SDN controller depends on accumulative knowledge to build statistical information and its perception about the ongoing traffic.

Depending on the above, the proposed IDS employs these available approaches to monitor the network status. Figure 4.1 depicts the three components of the proposed IDS; the first component is the (*Inspector* device) that is located between the edge switches and the SDN controller. It prevents unauthenticated users from launching saturation attack towards the SDN controller. The second component (CNN component) and the third component (RNN component) work as parts of the SDN controller to analyze the ongoing traffic in the infrastructure layer and the OpenFlow channel traffic.

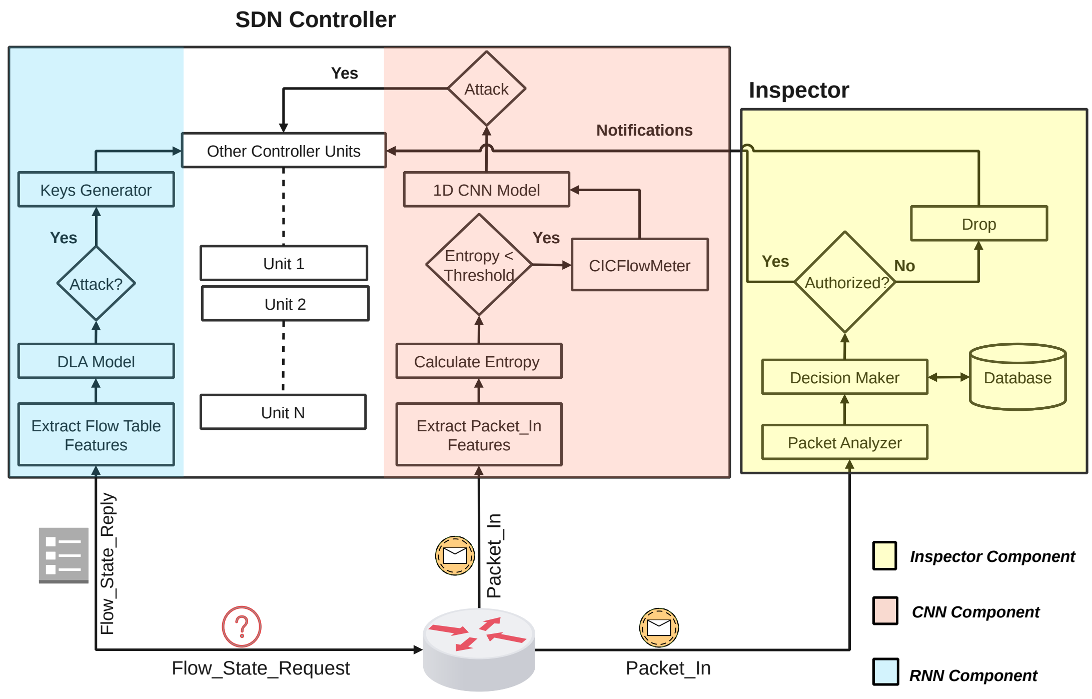


Figure 4.1. – The Proposed IDS Framework

Generally, the *Inspector* device handles networks that have a known number of users. The *Inspector* contains a database for all necessary information about the authenticated users.

The application layer could store the database from engineers and administrators, or the *Inspector* collects the users' information of the initial lifetime. The essential objective of using the *Inspector* is to prevent unauthenticated users from forcing the connected switch (edge switch) to submit *Packet_In* messages to the SDN controller. Therefore, when the user sends the packet for the first time, the connected switch forwards the packet to the *Inspector*. Thereafter, the *Inspector* verifies the user information compared to the stored information table (database). If there is a match, the *Packet_In* message is forwarded to the SDN controller to install a specific `Miss Table`. When the *Inspector* does not find any matching entry, the *Packet_In* message is dropped, and the *Inspector* notifies the SDN controller about the attack to block the attacker.

The second component (CNN component) monitors the *Packet_In* messages behavior by capturing the packet and calculating the entropy value based on the MAC addresses. Once the entropy value exceeds a predefined threshold, the One-Dimensional Convolutional Neural Network (1D-CNN) detects an injection attack in an SDN environment as the central part of the second component. Therefore, the 1D-CNN model reads the saved information about the network traffics (CSV file) to verify if there is a malicious attack or not. If there is an attack, the SDN controller installs a blocked rule. To prepare the CSV file, another tool known as *CICFlowMeter* that is responsible for extracting information for each bidirectional packet [CIC]. *CICFlowMeter* defines the first packet, no matter whether it is in the forward (source to destination) or backward (destination to source) direction. Consequently, the *CICFlowMeter* provides statistical information in a CSV format file. Each record in the file has more than 80 features that describe each network traffic such as duration, length of packets, number of packets, number of bytes, etc. Appendix C provides complete information about the features extracted by *CICFlowMeter*.

In the third component (RNN component), the SDN controller monitors the network performance by requesting the flow table from the OpenFlow switch and then analyzes the flow rules information using DLAs such as RNN, LSTM, and Gated Recurrent Unit (GRU). The RNN component periodically requests the flow table rules using the **Flow_State_Request** message. Thereby, the OpenFlow switch directly responds with the **Flow_State_Reply** message that contains the flow table. Once the component receives the replay message, some statistical equations extract 48 features to be inputs to the RNN models (Appendix B shows the selected 48 features). Hence, the model verifies if there is any malicious flow rule to define the connected Inport.

Finally, the third component has a dedicated unit to initiate the countermeasure procedure. Once the connected Inport is determined, the concept of IPSec protocol is applied to exchange the public keys between the SDN controller and the authenticated user who uses the connected Inport. After that, the SDN controller uses the received public key to generate a secret key. Then, the SDN controller installs flow rules containing the key as a specific field. At the same time, the user's machine uses the received public key to generate its encrypted key and add the key to the packet header. Eventually, the OpenFlow switch forwards the matched packets only.

According to the above illustration, we present the suggested IDS in five consistent papers [AS19b, AS19a, AS21, AFS, AS]. The followed methodology implements the ideas step by step. Each idea continues the previous work and introduces a solution for the main shortcoming. In detail, the remaining sections of this chapter clarify each step individually.

4.2 Inspector Device

The SDN concept supports dynamic and adaptive network management and minimizes the complexity of the forwarding elements [RR16]. Moreover, the SDN architecture enables the developer's innovations, policies, and applications to customize the SDN controller tasks. The OpenFlow protocol organizes the data exchange between the SDN controller and the OpenFlow switches. Thus, the SDN controller manages the traffic flows in the infrastructure layer by installing instructions into the switch's flow table.

The adversary exploits the OpenFlow channel to destroy the SDN controller by injecting a massive number of packets that overloads the SDN controller resources. Therefore, the adversary forges the packet header fields (e.g., IP or MAC address) to force the OpenFlow switch to direct the packets towards the SDN controller. After that, the SDN controller installs instructions into the switches, although the network does not need them. Also, the SDN controller establishes a wrong topology view because the crafted *Packet_In* Messages include untruthful addresses for non-existing devices. At the end, the adversary harms the network resources and destroys the network devices' functionalities.

Typically, the date exchanging between the SDN controller and the OpenFlow switches does not possess a method to prevent illegal access. Moreover, the SDN controller mostly installs temporary instructions to stop the attack. However, installing a new instruction for each new address is not efficient since the SDN switch has limited memory (TCAM stores between 1500 to 3000 entries) [PDFN17], [Oped] [ASS18].

4.2.1 Using Inspector Device to Stop Packet Injection Attack in SDN

This subsection explains the *Inspector*, while subsection 5.2.1 presents the simulation setup and the relevant results.

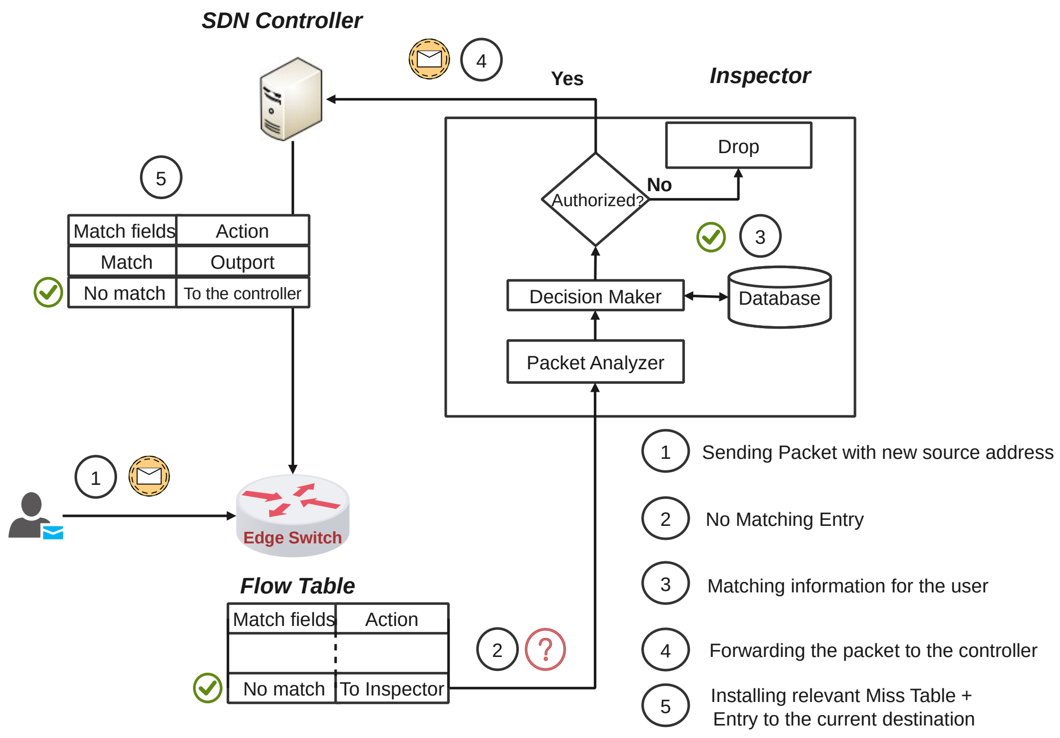


Figure 4.2. – Inspector Role

Figure 4.2 shows the proposed solution that adds a new hardware device verifying if the packet's source address has the authentication to send a *Packet_In* message to the SDN controller. The new hardware component (the *Inspector*) is an isolated machine that receives the *Packet_In* from the edge switches. Moreover, the *Inspector* solution considers that the complex network logic should exist only at the edge switches (ingress and egress switches). In contrast, the core switches should be kept as simple as possible. The edge switch is responsible for complex network services, such as network security, while the core switch provides basic packet forwarding. Forwarding elements are different in core compared to the edge switches because the core switch is not always required to use end-host addresses for forwarding [CHC18].

The *Inspector* is equipped with a database to store the users' information(e.g., Data Path Identifier of the connected edge switch (DPID), source addresses, and Inport number). It Additionally has a CPU to execute a comparison algorithm. The database enables the CPU to verify the user's information in the packet header. When the user sends a packet for the first time, the OpenFlow forwards the packet to the *Inspector* due to the packet contains a new source address. Thereby, the *Inspector* inspects the packet header to ensure that the user has the authentication to use the network resources.

This way, the *Inspector* looks up to match the packet header in the stored database. It decides whether the packet has authentication to access the SDN controller or not. Then, it notifies the SDN controller. Finally, the SDN controller installs a specific `Miss Table` containing the new source address. So, the other packets will use the OpenFlow channel directly. Thus, the proposed *Inspector* distributes the packet load between the SDN controller and the *Inspector* , especially during packet injection attacks. In contrast, when the *Inspector* does not find a matching entry in the database, it drops the packet. It then notifies the SDN controller about the attack.

4.2.2 Time Complexity of the Inspector

To calculate the time complexity T that the *Inspector* needs to respond to Packet_In. The assumption considers all devices and processes that transfer the Packet_In message to the SDN controller when the source address is new. So, the switches' number (S) is the summation of all switches on the path, where each switch is either core switch (Sc) or edge switch (Se). That means

$$S = Sc + Se. \tag{4.1}$$

The *Inspector* directly connects to the edge switch, which means the T only considers the edge switch time. In the *Inspector* , the time complexity to compare one *Packet_In* to the stored database containing n records is $\mathcal{O}(n)$ at the worst because the *Inspector* compares the *Packet_In* information to all the records stored in the database. Afterward, it sends the *Packet_In* message to the SDN controller using the OpenFlow Channel (of). Otherwise, the *Inspector* drops the *Packet_In* message

$$Inspector(n) = \mathcal{O}(n) \tag{4.2}$$

$$\mathcal{O}(of) = \begin{cases} 0, & \text{if an attack} \\ \mathcal{O}(of), & \text{if matching} \end{cases} \tag{4.3}$$

To conclude,

$$T(n) = Se + Inspector(n) + \mathcal{O}(of). \tag{4.4}$$

However, the OpenFlow channel uses TCP to transfer the data between the switch and the controller [Opef] [Opea]. As a result of that, the time to use the OpenFlow channel is mainly a constant time or ineffective.

$$\mathcal{O}(of) = \mathcal{O}(1). \tag{4.5}$$

In addition, the consumed time to switch the packet out of a switch is near to null (in ns) [SB16].

$$S \simeq null. \tag{4.6}$$

In the end, the consumed time depends on the comparison process in the *Inspector* .

$$T(n) = Inspector(n) \qquad \Longrightarrow \; T(n) = \mathcal{O}(n). \tag{4.7}$$

Generally, this extra complexity time affects as long as there is a new source address only. In two cases, the OpenFlow switch forwards the packet to the *Inspector* , either for the first time or the relevant `Miss Table` is deleted by timeout. In contrast, the switch forwards the packet to the SDN controller directly when the source address passed the *Inspector* before. Therefore, the complexity of our approach does not influence the network performance when the user obtained authentication to access the network resources.

4.2.3 Isolating the Inspector in case of a Malicious Attack

Unlike the traditional network structure, the physical separation between the infrastructure and control layers grants the researchers more ability to control the network operations. The researchers currently modify and assign new control layer components according to the network needs [DGL+19] [HJT+20]. The SDN controller is the network's brain that manages the network based on the coming instructions from the application layer. Hence, the intruder might exploit the Northbound API to harm the control layer's operations and disrupt its components.

Furthermore, the intruder pursues to control (hacking) a network device such as a controller, switch, or host. Then, the attack uses the device as a vulnerable point to launch malicious behavior to control the entire network or change the entries in one of the switches forwarding tables [AMK+18].

The *Inspector* device is an external hardware unit monitoring the incoming *Packet_In* messages to the SDN controller. Also, the *Inspector* is realized by the SDN application layer. That means there is a possibility to be hacked and controlled by the intruder. Thus, the *Inspector* could introduce a new vulnerability, mainly when the malicious host exploits to start up an attack on one of the infrastructure or control layer resources. Therefore, the SDN controller is extended with more functionalities to (1) monitor the *Inspector* performance, (2) update the external database, and (3) isolate the *Inspector* once the SDN controller notices an odd number of packets comes through a particular Inport [AS19a].

Mostly, when the attacker cannot corrupt the control layer, the attack is directed towards other users, and network servers (e.g., web server applications) [HTY$^+$19]. Consequently, the targeted server throughput becomes very low and the service is unavailable for other users. For example, the HTTP protocol makes the service available for web server users. Therefore, the attacker triggers malicious HTTP requests to make the server resources busy. This kind of attack is easy to execute because the rented botnets can attack the server. The botnets generate thousands of spoofed HTTP requests in a short period and with minimal bandwidth [HTY$^+$19].

The *Inspector* device should be located beside the SDN controller to ensure that the controller performance will not be affected [AS19a]. Furthermore, the *Inspector* connects to the edge switches directly to provide a rapid response to the edge switch requests. In addition, we assume that the complex network logic only exists at the edge switches. In contrast, the core switches should be kept as simple as possible [CHC18] [AS19b].

Based on the above, the *Inspector* contains several units to interact with the edge switches and the SDN controller, as is visible in Figure 4.3.

The *Inspector* uses East-West API to communicate with the SDN controller. The **Packet Analyzer** unit receives the *Packet_In* messages to obtain the packet header information (IP and MAC address, labels, port number, and DPID). After that, the **Decision Maker** unit implements a comparison to verify if the header information is included in the database. Besides, the **Controlling Messages** unit receives the SDN controller instructions and updates the database based on these instructions.

The connection between the controller and the *Inspector* has two channels: the first channel forwards the *Packet_In* messages to the SDN controller, while the second channel transfers the instructions from the SDN controller to update the database in the *Inspector* . Still, the SDN controller is responsible for managing the whole controlling tasks.

The *Inspector* would be the target when it is used to protect the SDN controller against DDoS attacks. Therefore, the work adds two units into the SDN controller for the second level of protection: **Traffic Data Collection** and **Statistics Extraction**. In SDN, the switches have a flow table and store traffic statistics [MOKKV19]. So, once the controller requests the switch's statistics, the switch sends them to the controller. As shown in Figure 4.3, the **Traffic Data Collection** unit periodically requests the statistical information from the edge switches over the OpenFlow channel. Also, the **Statistics Extraction** unit analyzes the edge switches statistics and compares the results to a threshold.

When the statistics extraction unit finds that the Inport's traffic is beyond a given threshold, the controller suspects that the *Inspector* or the OpenFlow switch is hacked. So, the SDN controller instructs the switch to delete all entries related to the suspected port and blocks the host. Moreover, the controller sends a message to the *Inspector* for updating the information related to the port in the database. But, if the same

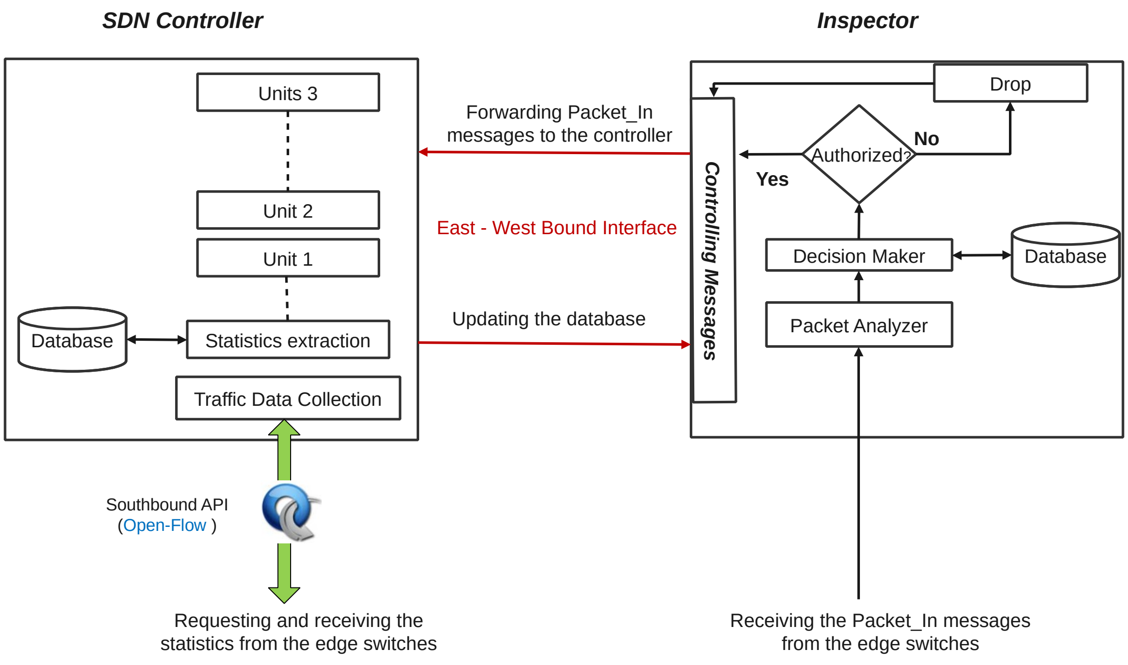

SDN Controller
Inspector
Units 3
Unit 2
Unit 1
Database
Statistics extraction
Traffic Data Collection
Forwarding Packet_In messages to the controller
East - West Bound Interface
Updating the database
Controlling Messages
Drop
Authorized?
No
Yes
Decision Maker
Database
Packet Analyzer
Southbound API (Open-Flow)
Requesting and receiving the statistics from the edge switches
Receiving the Packet_In messages from the edge switches

Figure 4.3. – Specifications of the Inspector and the Controller

Inport represents abnormal behavior after the interval, the SDN controller isolates the *Inspector* and works independently. As a result, the improved version protects the controller against hacking *Inspector* and a packet injection attack. The investigations of the improved *Inspector* are discussed in subsection 5.2.2

4.3 Convolutional Neural Network to Detect Control Layer Saturation Attack

4.3.1 Overview

The suggested *Inspector* device is effective when the user's information is known or there is an ability to enquire about the new user's information (e.g., cloud database). In contrast, some networks serve unlimited numbers and receive requests from different sub-networks through gateway devices such as the internet. As well, after repeating the suspect behavior from the same Inport, the SDN controller will dispense with the *Inspector*. Therefore, the SDN controller should have another component to handle the unlimited user's numbers or work as a second defense line after the *Inspector*.

Furthermore, subsection 4.2.3 exhibits the enhancement work on the *Inspector* performance, which uses a fixed threshold in the SDN controller to monitor the *Inspector*. Once the controller detects suspicious behavior, it starts the countermeasures. The CNN model offers a dynamic threshold to work with the *Inspector* instead of the fixed threshold. Therefore, this section employs the CNN model as the central part of the second component. Still, the results of the setup scenarios are presented in section 5.3.

Generally, the communication process between the controller and the switches is effective and enhances the flexibility and the programmability of the network. But, it provides new vulnerability and security challenges. The adversary exploits the communication by crafting numerous packets with different features (e.g., IP and MAC addresses). Thus, the crafted packets will not match any installed flow entry. Consequently, the `Miss Table` will forward the unmatched packets to the controller. This attack is a type of DoS attack known as the saturation or injection attack. It consumes the channels' bandwidth and limited resources in both of the control and switches [DGLG17].

Furthermore, the legacy approaches for detecting DDoS attacks are either human intervention or IDSs. The human intervention includes access control and relatively high labor cost. The IDSs might not be capable of differentiating benign traffic from malicious traffic perfectly. Likewise, the IDSs might have a high false alarm rate unsuitable for handling the SDN injection attack. Besides, with the scaling up of the internet or any other network type, the possibility of launching a malicious attack increases. That leads to destroying the performance of the network's components such as end devices, servers, mobile devices, electronic systems, and even the transmitted data [AS21] [WL20].

Therefore, the network administrators should develop an IDS to protect their networks. As well, the IDSs must efficiently classify the network traffic flows as benign or malicious by extracting and analyzing the traffic features based on predefined rules [EJNJ21]. Nevertheless, the attackers always enhance their methods to defeat the IDSs rules and generate traffic with the same benign traffic characteristics. As a result, the used IDS might not detect new malicious traffic.

Typically, MLAs and DLAs are efficient in understanding and classifying activities due to their features and many IDSs patterns used them. However, MLAs are mostly shallow learning where the algorithms have predefined characteristics for benign or malicious traffic. Furthermore, MLAs depend on the linear conduct of the data to have a perfect basis for classification. So, MLAs do not perfectly work with non-linearity behavior which leads to low accuracy. On the contrary, DLAs are the ideal solution presenting high accuracy for traffic classification. DLAs can extract the features and classify the traffics without previous knowledge or identifying the traffic behavior before executing the DLA. The 1D-CNN is a part of the second component in the IDS framework. The second component consists of the entropy calculation and CNN unit to detect an injection attack in an SDN environment as Figure 4.5, Figure 4.6, and Figure 4.7 declare. Also, The following subsection 4.3.2 explains the second component in detail.

4.3.2 One-Dimensional Convolutional Neural Network Model

CNN is a popular DLA for image processing applications. In addition, it is proving its success compared to other application areas that rely on sequential data such as audio, time series, and natural language processing. The convolution term is a mathematical equation working to combine two functions to produce a third function that aims to merge two sets of information [DSB21]. The CNN functions are in three different types: First type is one-dimensional convolutional is primarily utilized in sequential input, such as text or audio. Second type is two-dimensional convolutional is used where the input is an image. Third type is three-dimensional convolutional is applied to three-dimensional medical imaging or detecting.

In the suggested framework, the 1D-CNN is applied to time-series information describing a traffic flow at a specific port, device, or application. Figure 4.4 depicts the 1D-CNN methodology. In the first step, the traffic flow contains many packets creating the flow's

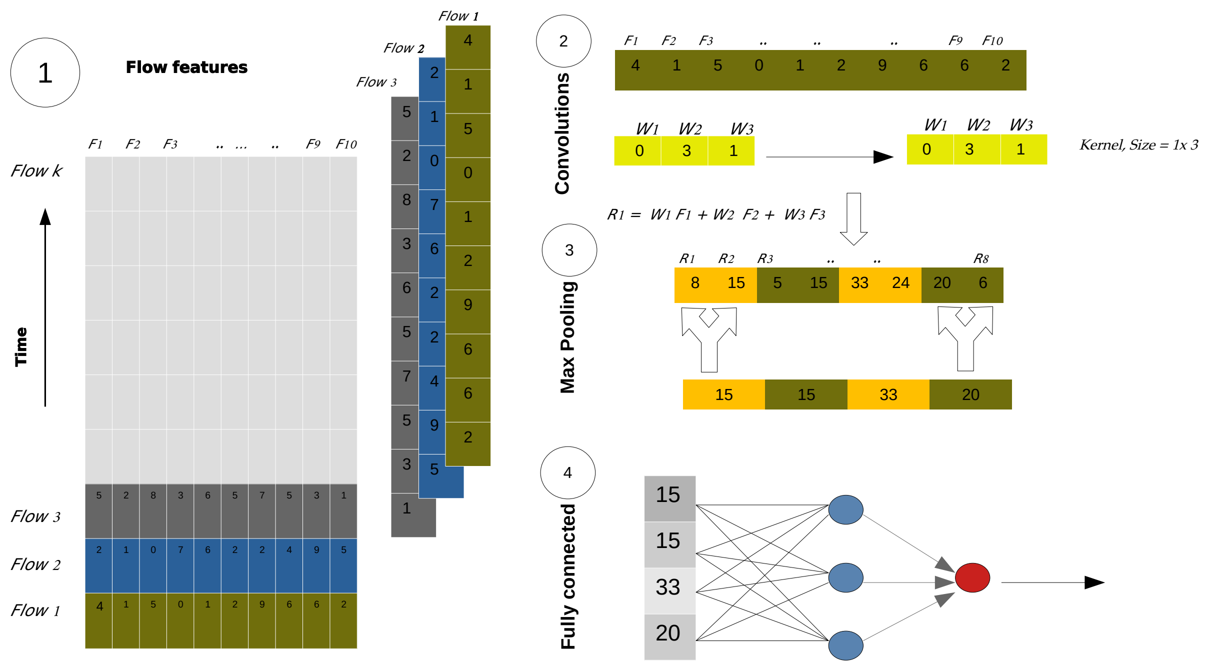


Figure 4.4. – One-Dimension convolutional Neural Network Technique

features. Every feature is an input value for the 1D-CNN algorithm. In the second step, the 1D-CNN scans the values with a kernel size filter $(A \times B)$ where A is the height and B is the width. The filter slides over the feature values to calculate the dot product between the filter and the features' parts meeting the filter $(A \times B)$. In the third step, the pooling layer uses another filter, obtaining either the maximal value or the average value (The second component selects the maximal value). Consequently, The filter decreases the input size and the computational power required to process the data through dimensional reduction [YNDT18]. Also, the most influential features are defined, which effectively maintains the model training [EJNJ21].

In fourth step, the outcome is either used as input for another convolutional and pooling layer or distributed to all neural nodes in a fully connected layer. The fully connected layer is a perfect method for treating non-linear values and learning a possibly non-linear function in that space. In the fully connected layer, each input's value has a percent wight. For example, if the value $(X = 20)$ and the percent weight is 20%, the real value is $(20 * 0.2 = 4)$. Nevertheless, the 1D-CNN flattens the result of the last pooling layer into a column vector to feed all neural nodes in the fully connected layer. Thereby, the neural node obtains the value's summation and uses the result (x) in the used equation based on trial (*ReLU* (rectified linear unit)) in Equation 4.8).

$$f(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ x, & \text{if } x > 0. \end{cases} \tag{4.8}$$

In the end, the output layer classifies the flow as benign or malicious according to the *Sigmoid* classification technique as stated in the equation 4.9.

$$Sigmoid(x) = \frac{1}{1+e^{-x}}. \quad Where\ e = 2.71828. \tag{4.9}$$

Moreover, the final results support a feed-forward neural network and back-propagation is applied to every iteration in the training time. The 1D-CNN model compares the final result to the correct value. Then, the difference between them is sent back to the neural nodes (output and fully connected layers) to adjust the percent weight of the inputs. Over a series of epochs, the model can distinguish diverse network flow traffics [DSB21].

The proposed 1D-CNN architecture comprises two convolutional layers with 32 and 16 filters. Each filter has a size of 1x3 and a stride of 1. The same padding is used in the convolutional layers to make the output the same as the input (adding zeros as inputs when the filter does not match enough inputs' number). Each convolutional layer is followed by a max-pooling layer with a size of 1x2. After that, the model uses two fully connected layers with neural nodes equal to 16 and 8, respectively. The final layer is one neural node that applies the *Sigmoid* function classifying the network traffic into benign (0) or malicious traffic (1). Furthermore, the 1D-CNN model uses a Nadam optimizer and a mean squared error (MSE). The hyper-parameter configuration is 100, 10, and 0.001 for the batch size, epoch, and learning rate. At the same time, the output layer uses binary cross-entropy as the loss function. In contrast, all layers except the last layer use the non-linear *ReLU* as the activation function.

To investigate the suitability for the SDN environment, the 1D-CNN model is applied as a unit of the second component. However, using the model for every packet or traffic is not acceptable because of the high computational complexity [WL20]. Therefore, three units are required before the SDN controller decides to mitigate any malicious attack. Figure 4.5, Figure 4.6, and Figure 4.7 explains the sequence of their performance in case of attack, suspicious, and normal behavior:

1. **CICFlowMeter Unit:** CICFlowMeter is a network tool that analyzes all bidirectional packets to generate specific information about the network traffic flow for a

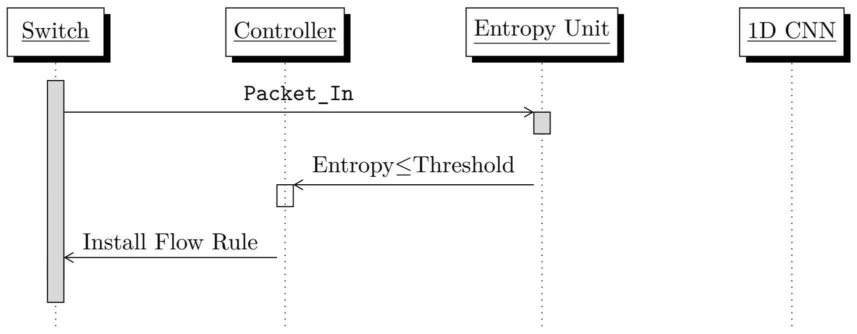


Figure 4.5. – Normal Conditions

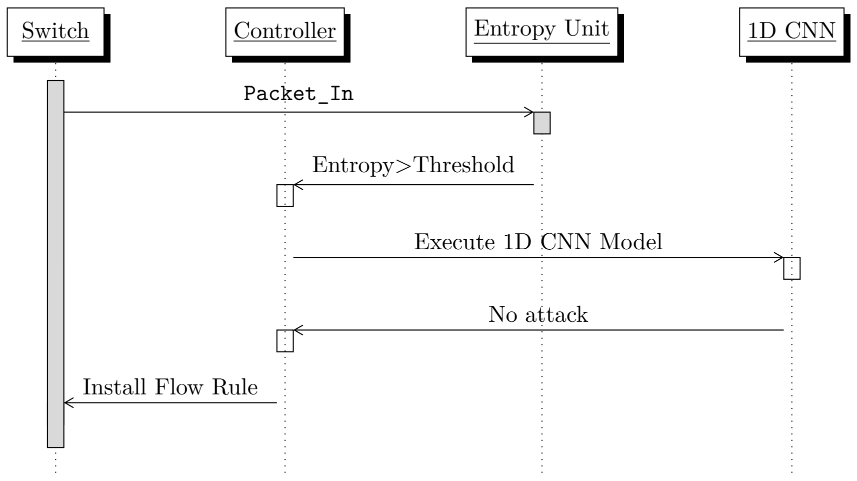


Figure 4.6. – Suspicious Behavior

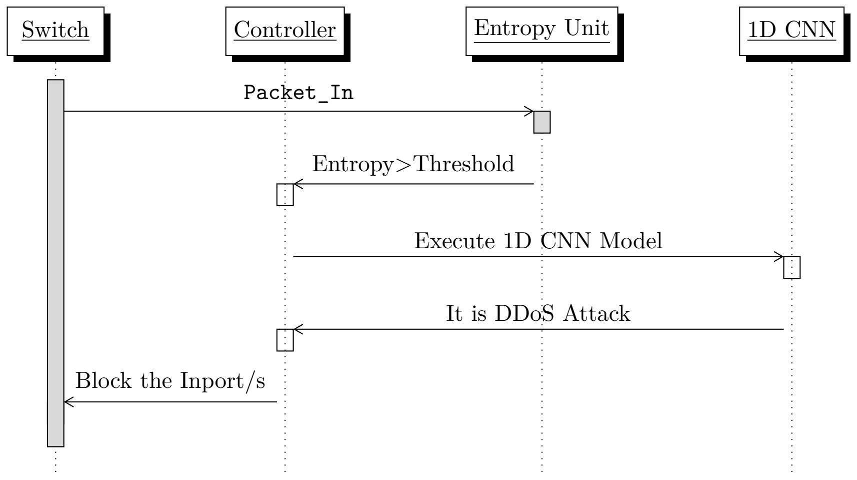


Figure 4.7. – Confirmed Attack

particular port, device, or network application. The CICFlowMeter captures the first packet of the flow, whether the packet is in the forward (source to destination) or backward (destination to source) direction. In the second component, the CICFlowMeter version 3 [CIC] tool captures all bidirectional packets of the SDN controller port number 6653. Thus, it provides statistical information in a CSV format file with more than 80 features for each network traffic. For instance, duration, length of packets, number of packets, number of bytes, etc. Appendix C shows all the extracted features [LDGMG17, fC19].

2. **Entropy Information Unit:** The entropy unit monitors the randomness of the *Packet_In* messages. In the injection attack, the adversary floods the SDN controller or the network with false header packet information (e.g., source or destination addresses). Once the adversary launches the attack, the *Miss Table* entry submits *Packet_In* messages through the OpenFlow channel to the SDN controller. As a result, the randomness value becomes higher than the predefined threshold, which signifies suspicious behavior. However, a flash crowd situation mostly behaves like malicious packets (e.g., packets per second, bits per second, and flows per second). Hence, the entropy information is not sufficient to identify whether there is an attack or not [WL20, CLMR19]. The entropy calculation is real-time capable and treats different traffic flows with a low computing overhead [XWX20]. Once the SDN controller receives the *Packet_In* message, the sensitive features are extracted and stored in the entropy window buffer. When the window is full, the unit calculates the randomness. If the entropy threshold is broken, the 1D-CNN unit is transferred from silent into the active state (see Figure 4.6).

   Generally, two components are essential to detect the DDoS using entropy, (a) the window size and (b) the entropy threshold value. Therefore, when the new *Packet_-In* messages reach the unit, the entropy window caches the sensitive features and the Inport information. Thereby, the entropy units value calculates the entropy as long as the window reaches the limit (This work uses 100 *Packet_In* messages based on the trials compared to 30, 50, 150, and 200). Entropy is calculated after this limit to measure the randomness in receiving packets. Equation 4.10 shows a window W, where f represents its frequency of occurrence. The entropy unit counts the repeated appearances for every single sensitive feature ϕ. It then calculates the occurrence's probability $P(\phi)$ based on the entropy window size W_{size} (where n is the last sensitive features and i is the number of the sensitive feature) like the following:

$$W = (\phi_1, f_1), (\phi_2, f_2), (\phi_3, f_3), ..., (\phi_n, f_n) \tag{4.10}$$

$$P(\phi_i) = f_i / W_{size} \tag{4.11}$$

and the Entropy value E of the full window is calculated in equation 4.12:

$$E = \sum_{i=1}^{n} -P(\phi_i) log_2 P(\phi_i) \tag{4.12}$$

Once the entropy threshold is violated, the 1D-CNN finds an attack in the last 50 flows of information (50 is based on the trial). The controller determines the malicious ports by calculating the port probability ($P(\phi)$) in the entropy window. Hence, the port of the highest $P(\phi)$ is blocked. The entropy value is recalculated by dividing the blocked port probability by its entropy slots in the threshold window. If the entropy value is less than the threshold or equal, the block port process stops. Otherwise, the controller will block the port of the second weight and so on.

3. **Dynamic Threshold Unit**: It calculates the dynamic threshold by considering the network traffic characteristics. The network traffic is always changeable and does not have the same conduct, and hence the threshold value should be adaptive. According to the algorithm 4.1, the size of the threshold window is changeable. If no suspicious *Packet_In* traffic, the SDN controller inserts the new entropy value into the threshold window. After that, the unit obtains the `variance` (V) for the window values twice; The first one is without the newly inserted entropy value ($Variance\_old$) and the second one is without the oldest entropy value in the threshold window ($Variance\_new$) as defined by equation 4.13 where n is the number of considered entropy values.

$$V = \sum_{i=1}^{n} (E - \mu)^2 / n. \tag{4.13}$$

As a rule, a minor `variance` value indicates that the newly inserted entropy is close to the `Mean` (μ). On the contrary, a high `variance` value indicates that the new entropy value is apart from μ. Additionally, the algorithm calculates the `quantity` (Q), which is the largest `variance` inside of a subset of all entropy values. Q aims to quantify the maximum V in the new window according to the following equation

$$Q = \frac{(\mu - S)(G - \mu)}{V} \tag{4.14}$$

Where S is the smallest entropy value, and G is the greatest entropy value in the threshold window. In addition, the unit uses the direct and inverse values to calculate the *DiffRatio* of the change. `Direct` measures the `Mean`'s differences between the current window and the last window. In contrast, the `Inverse` represents its inverse and measures the `Mean`'s differences between the previous window and the current window.

```
Input:
  • Mean of the current window values ($\mu_{new}$),
  • Mean of the previous window values ($\mu_{old}$),
  • Current dynamic window size (CWsize), smallest value (S), greatest value (G)
Result: Next Window Size (NWsize)
initialization
  $Variance_{new} = \sum_1^i (value_i - \mu_{new})^2/n$
  $Variance_{old} = \sum_1^i (value_i - \mu_{old})^2/n$
  $Quantity = (\mu_{new} - S) * (G - \mu_{new})/Variance_{new}$
  $Direct = Variance_{new}/Variance_{old}$
  $Inverse = Variance_{old}/Variance_{new}$
  $DiffRatio = \sqrt{(Direct - Inverse)^2}$
  if $DiffRatio > 1 + \beta$ then
      if $Variance_{new} > Variance_{old}$ then
          $NWSize = CWsize + \lfloor Quantity \rceil$
      else
          $NWSize = CWsize - \lfloor Quantity \rceil$
      end
end
return NWsize
```

Algorithm 4.1 – Dynamic Window Sizing

$$Direct = Variance_{new}/Variance_{old} \tag{4.15}$$

$$Inverse = Variance_{old}/Variance_{new} \tag{4.16}$$

$$DiffRatio = \sqrt{(direct - inverse)^2}. \tag{4.17}$$

Once $DiffRatio$ is greater than $(1+\beta)$ and the $Variance\_old$ greater than the $Variance\_new$, the algorithm increases the window size by rounding the `quantity` Q to the nearest integer. Nevertheless, when $DiffRatio$ is greater than $(1+\beta)$ and the $Variance\_old$ less than the $Variance\_new$, the algorithm reduces the window size by rounding the `quantity` to the nearest integer. The unit uses β to cancel excessive overhead, which equals 5% and represents a 95% confidence interval.

As is depicted in Figure 4.8, the unit uses the standard normal distribution to calculate the entropy threshold. The work follows the coverage theorem of normal distribution (what is well known as the empirical rule, also referred to as the three-sigma rule or 68-95-99.7 rule) [CLMR19]. The standard deviation (σ) represents 68% of the normal distribution values in the area between $\mu - \sigma$ and $\sigma + \mu$. Also, 95% of the values are in the area between $\mu - 2\sigma$ and $2\sigma + \mu$. However, 99.7% of the values lie in the area between $\mu - 3\sigma$ and $3\sigma + \mu$. Therefore, the SDN controller has all entropy values in the threshold window after a specific time. The unit calculates the Mean of the threshold window (μ) to obtain the value of the σ according to equation 4.18. In the end, the next point in the threshold is calculated by equation 4.19.

$$\sigma = \sqrt{\frac{\sum(E-\mu)^2}{CWsize}} \tag{4.18}$$

$$Threshold = \mu - 3\sigma \tag{4.19}$$

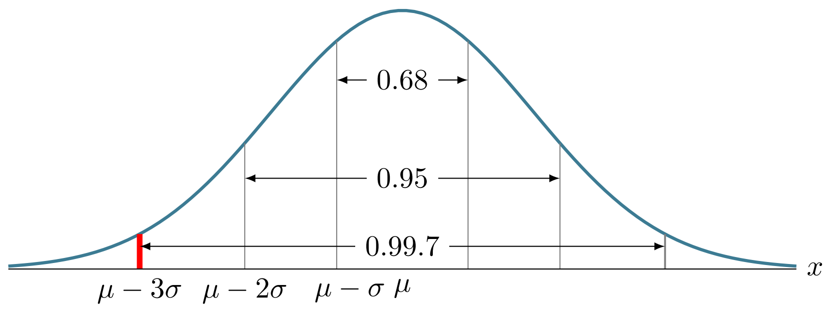


Figure 4.8. – 3-Sigma Rule

4.4 Toward Applying IPSec Protocol to Counter DDoS Attacks

The current networks are more complex than the networks three decades ago because of the massive number of end-users, connected devices, and modern applications. Therefore, the SDN enhances the legacy networking methods to respond to these changing requests [GBCMVVLV20].

Additionally, the malicious user exploits the available devices, such as advanced PCs or Botnets, to launch attacks towards different points in the network. For instance, the attacker transmits many messages to a specific address to exhaust its resources, such as link bandwidth, CPU usage, memory, etc.

In the previous sections, the IDS counters the DDoS attacks against the SDN controller. However, this section shows a novel idea to apply the IPSec protocol concept against the DDoS attack in SDN. At the same time, the experiment's results are presented in section 5.4.

The DDoS attack is a significant challenge when the attacker falsifies the IP and MAC addresses simultaneously. That means the attacker adopts an authenticated user role to exploit the pre-installed entries. This way, the attacker can target other end users, servers, or other network resources, which is known as a freeloading attack [PCK16]. The freeloading attack is possible in all communication network environments, especially in wireless and local area networks.

In such a freeloading attack, the malicious user eavesdrops on the transmitted packets using the promiscuous mode to obtain information about the traffic flows, sender, receiver, and route entries in the network. Subsequently, the attacker sends spoofed packets pretending to be benign. So, the attacker exploits the available resources illegally and exhausts the available resources and services in the infrastructure layer [CLMR19].

The attacker will direct the DDoS attack to the infrastructure layer in the SDN environment when the attack failed to attack the SDN controller. This section proposes an initial and novel approach to protect the SDN resources against freeloading and controller saturation attacks. Moreover, the solution introduces a unique method to distinguish between benign users and the attacker. The solution calculates a threshold for the expected amount of transmitted packets based on the Exponential Weighted Moving Average (EWMA). Furthermore, the solution adapts the IPSec concept between the end-users and edge switches in the SDN. Thus, the end-user adds a metadata field in the packet header as a signature during the DDoS attack. Moreover, the end-users and the controllers have a predefined algorithm for exchanging public keys, which assists both of them in generating secret keys for the signature. Ultimately, the switch inspects the secret key and then matches it to a specific field in the flow rule to differentiate between forged and correct packets.

4.4.1 Background: IP Security Protocol

The IP Security Protocol (IPSec) aims to secure and encrypt network traffic between the source and the destination over an unsecured network (e.g., two hosts, two gateways, or host - gateway). Hence, unauthorized users or applications cannot access or understand private data transmitted over the network. Also, IPSec protects IP packets against overhearing and enhances the defense technique against network attacks [KK05].

The unauthorized user or application can access or intercept the transmitted data as long as the two peer nodes exchanging data without encryption. When the user connects to a local or corporate network over the internet, the traffic is accessible for many unauthenticated users. Therefore, configuring IPSec on both communication sides makes the transferring date more secure. To implement IPSec, the sender uses an algorithm encrypting the data before sending; that means the transmitted data can only be understandable by an authorized user who knows how to decrypt the data. Moreover, the IPSec adds a signature to the transmitted data to ensure the data source [LMMLPG19] [HSI$^+$21].

Permanently, IPSec works based on two security methods:

1. **The Authentication Header (AH)**: which ensures the data's source (authentication/ origin/integrity). The AH ensures that the packet's source, payload, and header were not modified before reaching the destination. The equipped system with IPSec adds a new header between the IP header and before an upper layer security protocol or before the payload, which could be encrypted data according to IPSec [ZZ20]. AH contains several fields that are used or ignored due to the process need, such as next header, payload length, security parameters index, sequence number, and integrity check value. Finally, the packet could be sent by encapsulating the entire packet with AH (Tunnel Mode) or inserting AH between the IP header and the payload (Transport Mode) as the following :

AH in Transport Mode

IP	AH Hedear	TCP	Payload

2. **The Encapsulation Security Payload (ESP)**: offers data encryption and decryption. It encapsulates the entire packet between ESP's header and trailer except for the IP header (Transport Mode) or encapsulates the whole packet by adding a new IP header on top of that (Tunnel Mode). Both sides of communication should have a way to negotiate the encryption method and private keys. Besides, AH could be implemented independently or works alongside ESP protocol together according to the network needs [ZZ20].

ESP in Transport Mode

IP	ESP Header	TCP	Payload	ESP Trailer	ESP Authentication

The network layer is the targeted layer to implement the IPSec, but this does not conflict to apply its concept with other layers if it is possible. Therefore, the administrators should configure the end-user with the IPSec.

4.4.2 Background: Diffie–Hellman Key Exchange

IPSec involves various technologies and encryption methods. The most crucial step is the key exchange, known as Internet Key Exchange (IKE). The peers must be equipped either by a secret signature or a pre-shared algorithm such as the Diffie–Hellman key exchange algorithm.

Diffie–Hellman securely enables both authenticated peers to establish and exchange keys. In addition, the Diffie–Hellman uses the pre-shared prime modulus and a generator with random bits from a pool. In the beginning, each peer produces a public key and then sends it to the second peer. Hence, the peers use the received public key to calculate a shared secret key using the pre-shared algorithm. The secret key enables to set up of a secure tunnel and negotiate the parameter of IKE phase 2 parameters.

The purpose of IKE phase 2 is to negotiate with IPSec security association to set up the IPSec tunnel. The second phase of IKE is to negotiate IPSec security association parameters protected by an existing IKE security association and periodically renegotiates IPSec Security Association to ensure more security. Optionally, if there is a need, the users might perform an additional Diffie-Hellman exchange. Moreover, in the IKE Phase 2 negotiations, peers could select between ESP and AH, where ESP only does the encryption. In other words, when data is sent with AH, the transmitted data is unencrypted [Eas21].

To make a better understanding, let's suppose that Host X and Host Y launch an exchanging secret key procedure using the Diffie–Hellman key exchange Algorithm.

1. Both peers agree to use the same parameters, P is a prime number, and G (also known as a generator) is an integer fewer than P.
2. Host X chooses a private key A, then sends a public key ($APK = G^A \bmod P$) to Host Y.
3. Host Y Chooses a private key B, then sends a public key ($BPK = G^B \bmod P$) to Host X.
4. Host X calculates the shared secret key $S = BPK^A \bmod P$
5. Host Y calculates the shared secret key $S = APK^B \bmod P$
6. Finally, X and Y has the same shared-secret key.

4.4.3 Background: Problem Statement

The adversary always develops new approaches to target the network resources. The adversary might be selfish and surreptitiously piggybacks on authenticated user's network resources to reduce costs and even obtain free services. In addition, the adversary might be malicious and damages the target, such as DoS or DDoS attacks on remote servers, malware propagation into the network, spam, exploitation of remote hosts, etc [PCK16].

The adversary exploits the controller intervention in the SDN environment and starts launching many packets with different addresses. In other words, the adversary firstly sniffs neighbor information and then uses neighbor IP addresses to launch a large amount of traffic at the target resources. Therefore, the attack is difficult to be detected because the malicious traffic does not pass through the SDN controller, the intelligent network part. However, the adversary could launch the attack in SDN using the active flow rules in the switch table. Also, the adversary can piggyback its packets on the existing flow rules used as benign hosts. Although all users pay to upload and download data, the adversary obtains a freeloading for its malicious data.

This type is a freeloading attack that leads to consuming network resources, e.g., switches and links. The benign user has to pay more than its actual utilization because the attacker spoofs the benign user source addresses. Besides, once the attacker floods the network with high data-rate traffic, it fills the OpenFlow switch buffers, increases the loss packets, and minimizes the end-to-end delay.

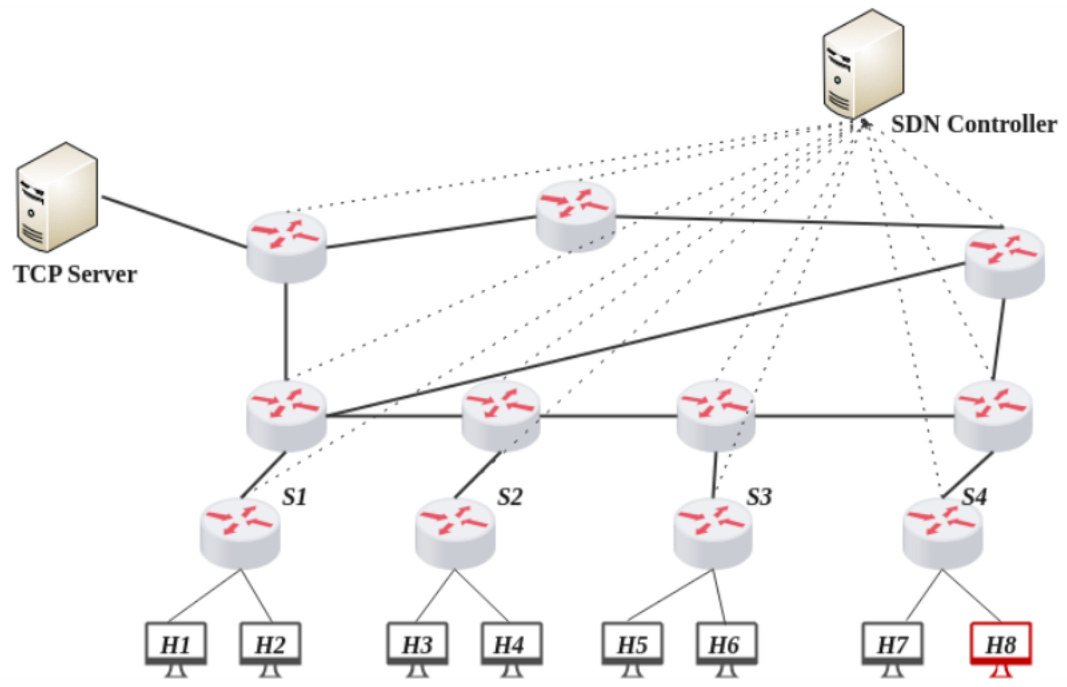


Figure 4.9. – Target SDN Network Topology

Inport	SRC/MAC	DST/MAC	SRC/IP	DST/IP	...	ACTIONS
1	00-00-00-00-00-07	00-00-00-00-00-01	10.0.0.7	10.0.0.1	...	2
1	00-00-00-00-00-07	00-00-00-00-00-05	10.0.0.7	10.0.0.5	...	2
1	00-00-00-00-00-07	00-00-00-00-00-03	10.0.0.7	10.0.0.3	...	2
1	00-00-00-00-00-08	00-00-00-00-00-03	10.0.0.8	10.0.0.3	...	2

Table 4.1. – Flow table of S4

Figure 4.9 clarifies the freeloading attack where Table 4.1 has a brief example of the forwarding table at OpenFlow switch (let's say it is $S4$). The $S4$ has flow rules installed by the SDN controller (see Table 4.1). Port 1 would forward all the packets received with source addresses of *H7* and destination addresses of *H3* to output port 2. Also, the OpenFlow switch forwards all packets received from port 1 with source addresses of *H7* and destination addresses of *H1* to port 2. The same result with *H5*. *H8* is under a malicious user control, and as per the entries in the second table, it can only send data to host 3.

Both *H7* and *H8* are connected to the same switch. In this scenario, the adversary (*H8*) crafts the addresses of *H7* since both of them are in the same local area network.

Consequently, *H8* exploits the spoofed addresses to launch malicious traffics to all destinations that *H7* uses through port 1. That means the adversary can obtain a free service, and *H7* pays for the service. The adversary would also be capable of launching a DoS attack toward all of *H1*, *H5*, and *H3* with bearing no legal consequences.

4.4.4 Proposed Countermeasure

The IDS framework applies a novel solution to distinguish between the malicious and benign users under the DoS, DDoS attack circumstances, to address the problems mentioned in 4.4.3 section. The IDS relies on the AH technique to provides the benign user a secret key (signature). Like this, the edge switch drops the malicious packets only. In this section, the solution considers UDP, ICMP, and TCP protocols and their differences. But, in the next section 4.5, the IDS proposes to enhance the solution to be compatible with the other IDS proposals.

4.4.4.1 Miss Table Architecture

The OpenFlow switch would forward no packet to the SDN controller as long as the adversary crafts packets to match active flow rules. Thereby, the controller cannot count the new SYN request or inspecting the packet attributes. Therefore, the edge switch uses two `Miss_Tables`. The first table monitors SYN requests, and the second table monitors the amount of the sent data.

Table 4.2. – SYN Table

Priority = 1	Match(Flags = SYN)	Action = Controller
Priority = 0	Match(Otherwise, anything)	Action = Table1

The SYN table 4.2 enables the SDN controller to monitor SYN requests. So, the edge switch sends the new SYN request to the SDN controller as a *Packet_In*. The controller then inspects the packet to decides the forwarding behavior. Finally, the controller installs a flow rule containing complete information about the source addresses and the Inport number.

If the same SYN request comes through the same Inport, the first table immediately forwards the request through the proper Outport. It then increases the related counter of the match field by 1. Every 3 seconds (the best time period based on the experiments used different suggestion periods), the SDN controller requests the flow states from the SYN table to analyze the flow rules statistics. The controller uses the match field and the corresponding packet number to define the TCP user's behavior. After that, the controller compares the TCP user's behavior to all other users in the network history.

Table 4.3. – The Second Table

Priority = 0	Match= (anything)	Action = Controller

The second table 4.3 acts as the `Miss_Table` to treat all packets coming from the SYN Table (the packets without SYN Flag). The new packet that does not match any flow rule, the `Miss_Table` entry forwards it to the controller as a *Packet_In*. Thus, the SDN controller makes a decision and installs a suitable entry to the second table. Furthermore, the SDN controller requests the flow states of the second table every round and monitors the amount of the sent data in the edge switches.

4.4.4.2 Adaptive Threshold Algorithm Based on EWMA

The EWMA statistically monitors the target's behavior during the last time. To do that, EWMA calculates multiple points of the averages in a particular way to provide less and less weight of the data because they are further removed in time [GD21].

In this work, the EWMA unit provides a pointer about the sent data during the previous lifetime. Therefore, The SDN controller has a threshold for each table in the edge switches. Hence, there is a threshold for SYN requests, and the second one monitors the sent data. When the user data exceeds the threshold, the unit issues an alarm about a suspicious user.

Let us assume X_i is the number of the SYN requests for a certain Inport. Also, the average value of all SYN requests in the previous rounds (from the beginning until now) is T_{t-1}. That means the threshold is obtained by the equation 4.20.

$$Threshold = (p + 1) * T_{t-1} \tag{4.20}$$

Where (t) is the round number, and the percentage parameter (p) observes any notable increase for the SYN requests or the sent data. Thereby, the controller alarms at round number t when.

$$X_i \geq (p + 1) * T_{t-1} \quad where, p > 0 \tag{4.21}$$

Typically, The controller uses two units to verifies the traffic's behavior. The first unit periodically requests the flow states (3 seconds in this work). The second unit receives the flow states and counts the sent packets for each Inport in both tables. Figure 4.10 shows a flowchart for the second unit, where the second unit has two values for the

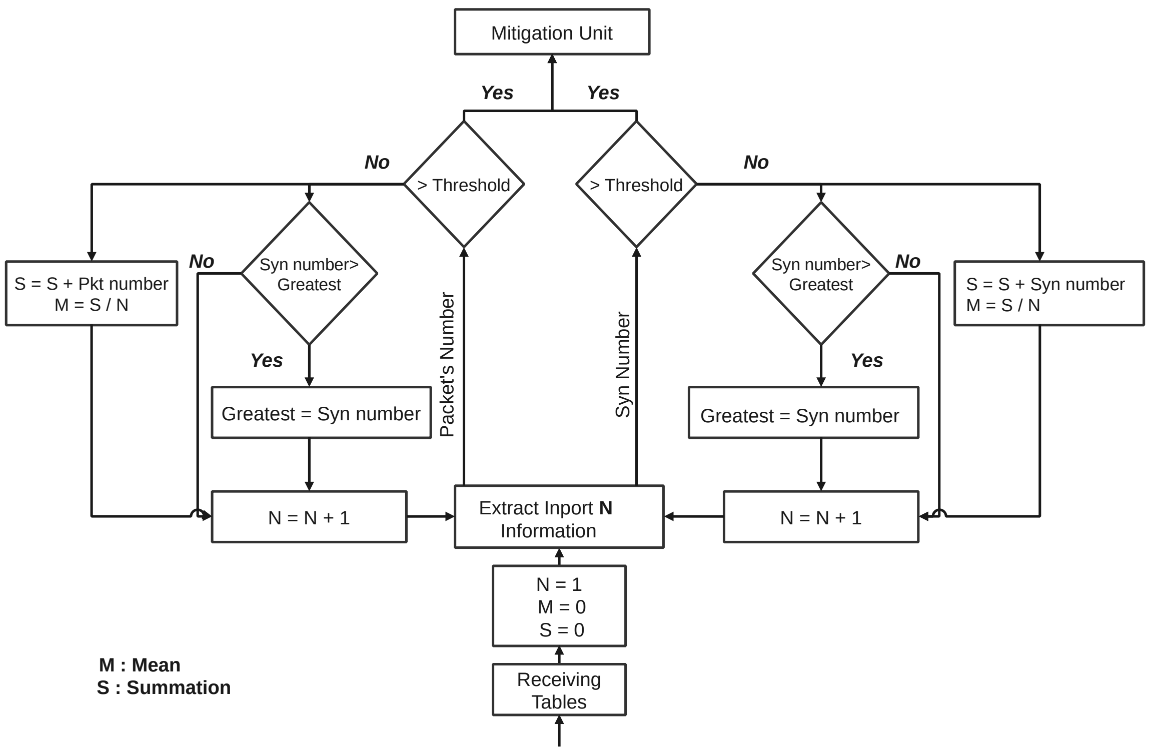


Figure 4.10. – Flow Chart of the Second Unit

same Inport. The first value counts the SYN packets using the SYN table. As well, the second value counts the sent packet using the second table. After that, the unit compares each number to the relevant threshold; if it is greater than the threshold, the unit alarms the controller about the suspect user. Otherwise, the unit considers the highest number for all flow tables as the X_i. It is worth mentioning the SDN controller has two thresholds for both SYN packets and the amount of data.

X_i acts one point on the timeline, and it is unreasonable to consider it like 50% in the new average value. Therefore, The new average value (T_t) can be calculated over time by the EWMA formula 4.22, which considers α (the value's percentage).

$$T_t = (\alpha * T_{t-1}) + ((1-\alpha) * X_t) \quad Where,\ 0 \geq \alpha \geq 1. \tag{4.22}$$

Furthermore, the network could be silent for some consecutive rounds. Therefore, the X_t value would be around zero, which leads to getting down the threshold less than the network's normal behavior. Therefore, the equation 4.23 adds an Adaptive baseline b.

$$Threshold = (p + 1) * T_{t-1} + b \tag{4.23}$$

Adaptive baseline b is the window average for the last round. Algorithm 4.1 in the end of subsection 4.3.2 describes the operation to predicates the Dynamic-Window Size when there is no suspicious user. The second unit calculates the Mean value (μ) for all flow states of the edge switches every round.

In other words, the controller periodically retrieves the number of sent SYN flags from the SYN table and the number of sent packets from the second table for every Inport. Thereby, the controller calculates μ for both tables to insert them to the previous results in their dynamic window.

Moreover, the variance (V) is calculated twice for all available values in the dynamic window. The first time is without the new value in the window ($Variance_{old}$), and the second time is without the oldest value ($Variance_{new}$).

$$V = \sum_{i=1}^{n} (value_i - \mu)^2 / n \tag{4.24}$$

n is the number of considered values. The small V indicates that the new added value is close to μ. If V is a high value, the added value is passing away from μ (Mean of the window values). While `quantity` (Q) is the biggest variance inside a subset of all values. The algorithm 4.1 calculates the Q to quantify the maximum V of the current Dynamic window and know the variation of the window size,

$$Q = (\mu - S) * (G - \mu)/V \tag{4.25}$$

S is the smallest value, and G is the greatest value in the targeted window. Moreover, the algorithm needs to obtain the direct and inverse values to calculate the ratio's difference of the change ($DiffRatio$). *direct* measures the mean changes between the current and last windows, and *inverse* represents its inverse and measures the mean changes between the last and current windows.

$$direct = Variance_{new}/Variance_{old} \tag{4.26}$$

$$inverse = Variance_{old}/Variance_{new} \tag{4.27}$$

If $DiffRatio$ greater than (1 + β) and the $Variance_{old}$ greater than $Variance_{new}$ increasing the window size by quantity. But, in case the $DiffRatio$ is greater than (1 + β) and the $Variance_{old}$ less than $Variance_{new}$, then decreasing the window size by quantity value.

$$DiffRatio = \sqrt{(direct - inverse)^2} \tag{4.28}$$

The algorithm uses β to cancel superfluous overhead, β equals 0.05, representing a 95% confidence interval. In other words, confidence interval relates to a boundary value for the population values. So, the difference between the current value and the mean of the last window is not significant at the 5% level.

4.4.4.3 Controller and User Countermeasure

The OpenFlow Protocol offers secure communication between the SDN controller and the forwarding nodes. This communication provides an excellent transporter to transfer the table information, network nodes status, and user requests for routing. Besides, the controller sends instructions to manage the network, and the OpenFlow protocol transfers the flow state from the table to the controller based on the controller's request. Also, the controller can define a particular flow entry by its attributes (priority, match, table number, etc.) [Nip21].

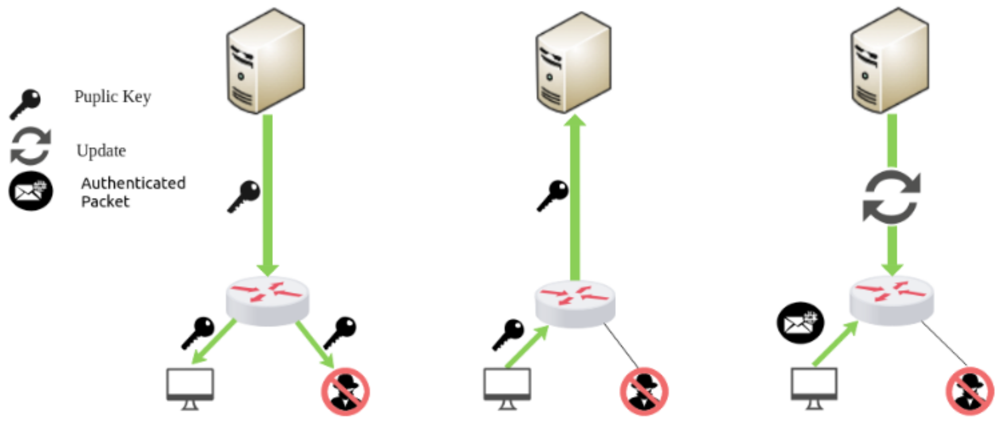


Figure 4.11. – Controller Mitigation Producer

In this proposed method, the controller periodically sends requests to all flow tables (SYN table and the second table) in the edge switches to obtain complete information about the flow state. Subsequently, the SDN controller counts the packets passed the Inports, compares the numbers to the threshold, adds the greatest value to the dynamic window values, and verifies the need to increase or decrease the window size. After that, the controller calculates the baseline.

The mitigation unit would be silent until one of the values exceeds the predefined threshold, triggering the controller to launch the mitigation procedure. Figure 4.11 depicts the mitigation procedure after the controller detects a suspicious user. Using IPSec protocol and Diffie-Hellman concepts, the controller calls the shared prime and generator numbers. Hence, the mitigation unit generates a public key and sends it to the edge switch, which forwards it to the user through the connected port. Afterward, the controller launches the mitigation by sending an instruction to the edge switch to delete all flow states related to the suspicious user and its connected Outport.

Finally, the controller installs two entries. The first one blocks any packet without key. The second one has the higher priority to forward the tagged packet with a key to the controller. Meanwhile, the controller would not answer any request to install a new flow for this user unless it receives a *Packet_In* that has the public key which used to generate a secret key. Thus, the controller uses the secret key to verifies from the *Packet_In* source by comparing it to the signature produced by the user.

Likewise, the user uses its prim and generator number to send another public key back to the controller. Furthermore, the user generates its encrypted key based the received public key from the controller and uses it as a signature according to the AH concept from the IPSec protocol. When the controller receives the first packet containing the signature, the controller installs the requested entry into the edge switch. The flow entry has higher priority and a metadata field to match the signature in the packet header. In addition, the entry removes the signature and forwards the packet to the proper Outport.

Eventually, the controller updates the previous two installed entries to forward the packet to the controller if the packet has the secret key or drops the packet that comes through the Outport without the secret key.

4.5 Recurrent Neural Networks to Classify Traffic

The SDN controller would be out of service when the received number of requests exceeds its abilities. Also, the benign user sometimes behaves similarly to a malicious user. Especially when the number of requests is increasing because of temporary reasons (e.g., free services on some servers), which is known as flash crowd [YWLW20] [CLMR19].

Thus, implementing robust security regulations is essential to protect the controller against DDoS attacks. Furthermore, the solution has to distinguish between malicious and benign users with a high accuracy that ensures the lowest false alarm rate and minimizes the associated computational cost as much as possible.

The section 4.1 presents the third component in the suggested IDS framework. The third component depends on the flow table to detect the network behavior. In case of an attack, the third component notifies the mitigation unit. Then, the mitigation unit exchanges the key with the users. Hence, the third component enables the OpenFlow switch to differentiate between the malicious and benign packets. However, section 4.4 presents how the third component handles the TCP, ICMP, and UDP packets.

In this section, the IDS framework applies RNN models as an enhanced solution to the third IDS component in section 4.4 to perform as the flow chart in Figure 4.12. Moreover, the RNN models can achieve high accuracy, detects more attack types, minimizes the number of flow tables, and cancel all complex statistical procedures. Therefore, in this

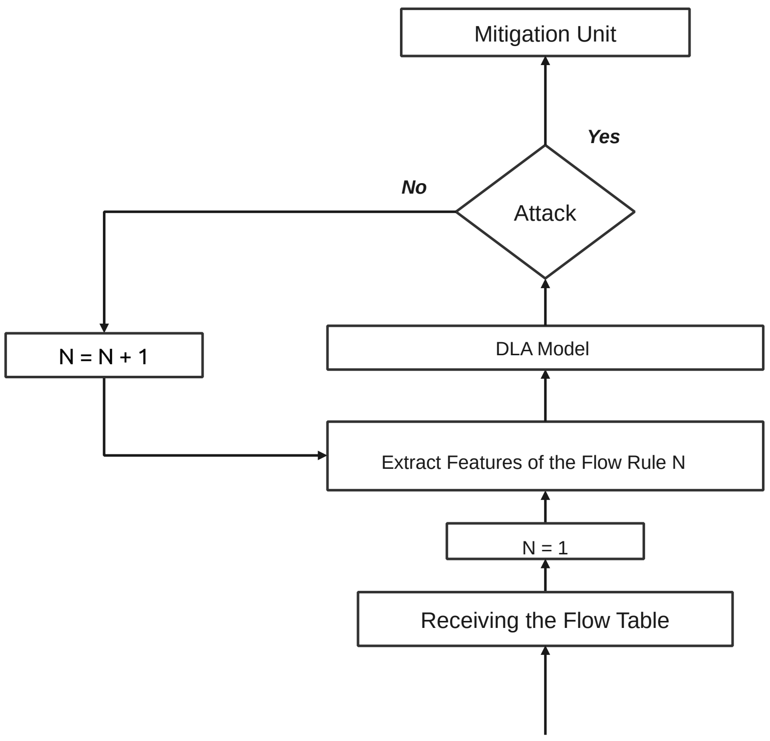


Figure 4.12. – Flow Chart of Third component to Detect the Malicious Behavior

section, the performance of RNN, LSTM, and GRU is explained for the IDS-based on flow states. The relevant implementation and results of this section are presented in section 5.5.

4.5.1 Recurrent Neural Networks

Different research depends on the RNN and development areas such as image processing, reading handwriting, and speech recognition. Also, RNN is a neural network algorithm designed to capture information from sequence points. Therefore, RNN can solve a problem that requires processing a sequence of data to feed-forward neural networks.

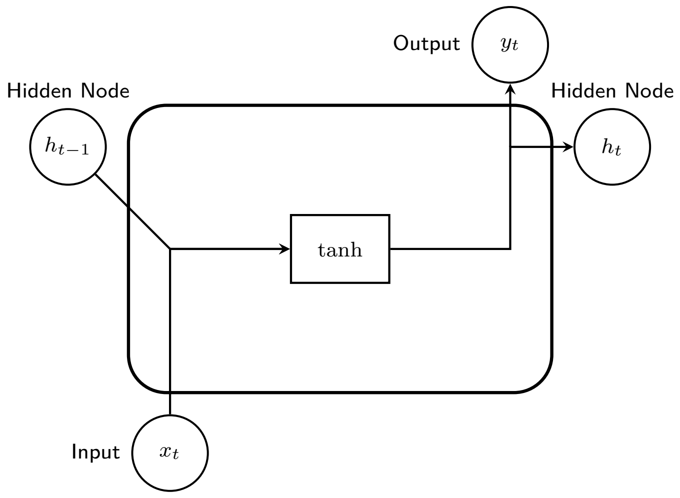


Figure 4.13. – Basic Architecture of an RNN Neuron

Furthermore, RNN has a recurrent structure because the output of one step is fed as an input to the next step [JFG+18]. For a better understanding, assume the input to the model over time t given by the vector $X = (x_1, x_2, x_3, ..x_t)$. Moreover, the neural nodes in the layers between the input and output nodes (hidden nodes) can be computed as the vector sequence $H = (h_1, h_2, h_3, ..h_t)$. The output nodes are considered as the vector sequence $Y = (y_1, y_2, y_3, ..y_t)$. The RNN model, as depicted in Figure 4.13, calculates the hidden node h_t at time t according to the following equation:

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h); \tag{4.29}$$

Where σ is the activation function, W are different weights, b_h is the bias and h_{t-1} is the hidden node at time $t-1$. Also, in the end, the output y_t at time t is defined in the next equation:

$$y_t = W_{hy}h_t + b_y \tag{4.30}$$

Finally, RNN enhances the output by learning from the treated data, which is known as back-propagation. Every time the RNN receives data, it produces an output, and then the output is compared to that of the desired result by a loss function. Repeatedly, the RNN adjusts the weights based on the difference between the output and the target result. However, initial weights are assigned at the beginning with the random values close to zero. When the RNN starts the back-propagation by multiplying a tiny number by the weight values, the result becomes less and less called the vanishing gradient problem. Therefore, LSTM and GRU appear to solve this problem.

4.5.2 Long Short-Term Memory

LSTM is an enhanced architecture of RNN used for deep learning applications, such as IDSs, speech recognition, and connected handwritten applications. Also, the LSTM architecture solves the vanishing gradient and learns from experience to classify a time series [TMM+18]. In general, an LSTM has a similar control flow as an RNN. But the differences are the operations inside the LSTM's cell. Figure 4.14 shows the architecture of LSTM's cell.

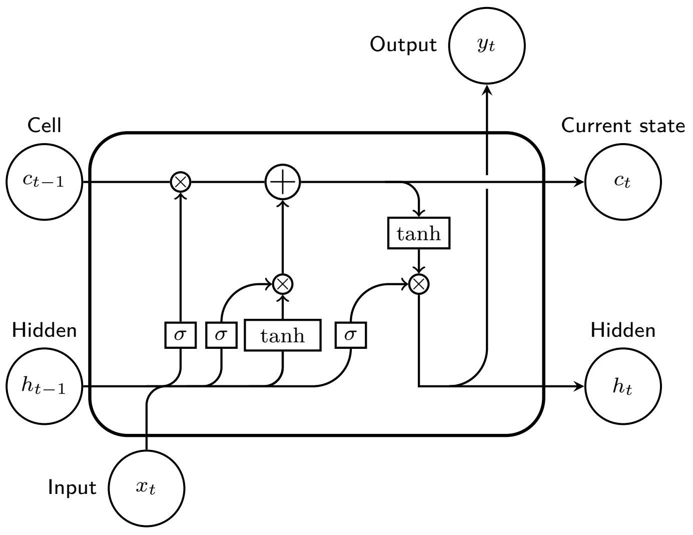


Figure 4.14. – Long Short Term Memory Cell

The LSTM's cell memorizes the values over arbitrary time intervals. Also, three gates adjust the data in the cell: an input gate, an output gate, and a forget gate. In general, the essential functionality of the LSTM cell is to decide which information should be deleted or memorized. The forget gate is responsible for information coming from a prior hidden state (h_{t-1}) and from the input (x_t). It combines them and applies the σ function to get values between 0 and 1. If the value is closer to 0, the forget gate would remove the information. Otherwise, the forget gate would pass the value to the next step to process with the previous cell state (c_{t-1}). Moreover, the result of ($h_{t-1} + x_t$) passes through another σ and tanh function respectively. Thereafter, the result of both σ and tanh would be multiplied. Eventually, the current state (c_t) is calculated by the next equation:

$$c_t = c_{t-1} \otimes f_t \oplus I_t \tag{4.31}$$

where, f_t is the forget gate

$$f_t = \sigma(W_f(x_t + h_{t-1})) \tag{4.32}$$

and I_t is the input gate, which consists of two equations:

$$i_{t1} = \sigma(W_{i2}(x_t + h_{t-1})) \tag{4.33}$$

$$i_{t2} = \tanh(W_{i1}(x_t + h_{t-1})) \tag{4.34}$$

With the previous two equations, we obtain the input gate I_t value as

$$I_t = i_{t1} \otimes i_{t2} \tag{4.35}$$

The output gate decides which information in the cell state should send to the network as input in the following time step. The output gate's activation is given by.

$$h_t = \sigma(W_o(x_t + h_{t-1})) \otimes \tanh(I_t) \tag{4.36}$$

LSTM solved the problem of the vanishing ingredient. Still, it is more complex than RNN because LSTM needs more resources and time to train and be ready for real-world applications.

4.5.3 Gated Recurrent Unit

GRU is another advanced type of RNN similar to LSTM. It needs less time to train because the gates' structure is simple compared to LSTM because it lacks an output gate.

GRU contains two *Sigmoid* gates and one hidden state. The computation can be summarized as:

$$h_t = (1 - z_t) \otimes h_{t-1} \oplus z_t \otimes i_t \tag{4.37}$$

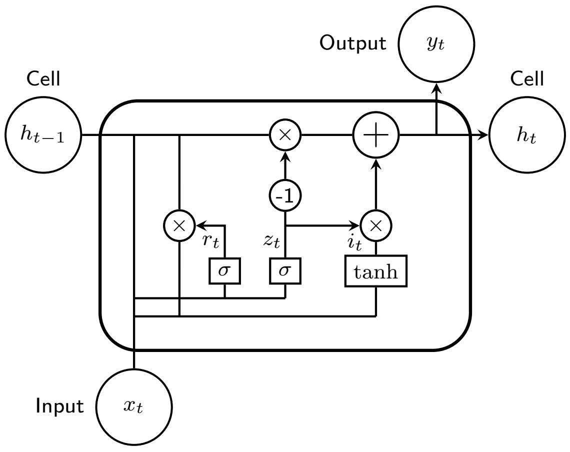


Figure 4.15. – Gated Recurrent Unit Cell

where z_t , r_t and i_t are obtained as the result of the next equations according to the depicted gates in Figure 4.15:

$$z_t = \sigma(W_z(h_{t-1} + x_t)) \tag{4.38}$$

$$r_t = \sigma(W_r(h_{t-1} + x_t)) \tag{4.39}$$

$$i_t = \tanh(W_i(r_t \otimes (h_{t-1} + x_t))) \tag{4.40}$$

Figure 4.15 shows the architectural details of a single GRU cell, which replaces the forget and input gates with an update gate i_t, adds a reset gate r_t in order to modify h_{t-1} and removes the internal memory of LSTM cell c_t.

In other words, the main differences between GRU and LSTM are the gates number and the maintenance of cell states. In contrast to GRU, LSTM has three gates (input, forget, and output) and maintains an internal memory cell state. Consequently, LSTM is more flexible but less efficient memory and time-wise. Although GRU and LSTM are perfect for resolving the vanishing gradient problem, both need to track long-term dependencies. Moreover, it is recommended to extensively train an LSTM initially because it has more parameters and is a bit more flexible. Nevertheless, in case there are no quantifiable differences in their performance, GRU would be better because it is simpler and more efficient [HOC+20].

4.6 Discussion

The flexibility and agility of the traditional networks are the essential goals of SDNs. Therefore, many researchers and projects evolve SDN to reach the optimal structure. Still, the SDN structure suffers from vulnerable aspects allowing the malicious attacks to threaten the network architecture in addition to the threats that are already existing in the traditional network.

The suggested IDS framework considers three methods that gather the traffic information: the OpenFlow channel, external device, and flow tables. Each method is the input for three different techniques detecting the DDoS attacks, as Table 4.4 demonstrates their objectives generally. Thus, the main feature of the suggested IDS is the flexibility, where the network's administration can depend or dispense with the IDS components according to the network requirements. Furthermore, the network's administration can easily insert any enhancement in the suggested IDS because each component independently works and acts as a second defense line for the other component.

This IDS employs the DLAs (CNN, RNN, LSTM, and GRU), which succeeded in many applications. The DLAs can analyze both training and test data and use any ongoing data to correct the understanding for the upcoming data. That way, the SDN controller is ideal for gathering traffic information and extracting features from the data without any human intervention. That offers a ready output to feed the used DLAs.

The IDS framework presents a new countermeasure relying on IP Security protocol. Through the IPSec protocol concept, the IDS framework offers a new approach as the first step-stone toward a full immunization by using IPSec protocol between the edge switches and end-users. Therefore, the Mitigation unit implements the AH method to differentiate between benign and attacker during the freeloading or DDoS attack.

However, the dump devices might work against the SDN objectives and minimize the network response. Some projects have started developing new approaches providing more abilities to the forwarding devices. For instant, BEBa-project [BEB21] implements the Finite State Machine (FSM) in the OpenFlow switch. Still, FSM is limited to change the flow entry from state to another and execute different action. FSM cannot compare simple computation processes, as it is dependent on the controller to change the entry fields. Initial projects begin adding more power to the OpenFlow switches to enable simple computations and memory operations [RBN$^+$19].

Finally, the edge switches act as the connecting point between the network and the end-users. Thus, the first defense line against the threats mostly comes from connected unauthenticated or hacked devices. Therefore, the edge switches should be more compatible and extended to run more computation processes.

Changing the entry fields depending on data analysis or traffic behavior would make the network more sensitive to changes and make better decisions. Furthermore, the small comparison could allow the switches to read the encryption data and rebuild it as the original or extend the area to apply other security algorithms and protocols.

The component	The Objective
Inspector Device	First component that handles the networks with a known number of users (e.g., local network). It prevents the malicious users from launching the saturation attack towards the SDN controller
CNN Component	Second component that handles the networks that serve an unlimited number of users. It might also act as the second defense line after the *Inspector* in the local networks and introduce an adaptive threshold instead of the fixed threshold in*Inspector* Deveice
RNN models	Third component that handles the inherited DoS/DDoS types from legacy networks. It also presents a new method to countermeasure by distinguishing between the malicious and the benign users

Table 4.4. – Overall Functions of the Proposed IDS

Chapter 5

Simulations and Experiments

This chapter describes the simulation environment, the modeled networks, and the conducted scenarios used to evaluate the proposed IDS framework by different components. Furthermore, it provides details regarding the network's typologies and setup. Besides, the chapter grants information about the hardware and software tools used to create the SDN typologies and simulate the DDoS attacks behavior in SDN.

5.1 Simulation Environments

The Mininet emulator and Ryu controller successfully conducted the simulations. Firstly, Mininet is a Linux-based network emulator that allows the network engineer to create a realistic virtual SDN. Mininet creates virtual hosts, switches, controllers, and links. In addition, it supports the OpenFlow protocol and runs a real kernel and application codes on a machine. Mininet uses Command Line Interface (CLI) and API to interact with the simulated network. In addition to the Miniedit capabilities that create and configure network typologies, and allow to create, interact, and customize prototypes for SDN applications [Min21]. Mininet offers many advantages as following :

1. Easy-to-learn and inexpensive network testbed.
2. Easy installation, either with simple commands or using pre-build virtual machine.
3. Provides a Python API.
4. Support integration with real environments.
5. Easy integration with known SDN controllers.

6. Support inter-controllers communications between different controllers [dOSSP14].

Secondly, Ryu Controller is entirely a python-based and open-source SDN controller. It supports the OpenFlow protocol and all available SDN controllers. Moreover, Ryu exchanges messages with OpenFlow switches to manage the infrastructure layer [Nip21]. In addition to the Ryu controller and Mininet Emulator, the investigation processes need to launch various traffic types and craft the packet according to the scenario requirements. Therefore, the experiments use several available tools that are summarized as following:

1. **Scapy**: is an excellent interactive packet manipulation software tool. It can forge or decode packets for different protocols before transmitting them through the wire. Also, Scapy captures the packets, matches requests and replies, and much more. It can easily handle most classical tasks; Scapy can scan, traceroute, probes, tests unit, attacks, or network discovery [Sca].

2. **Hping, Hping3**: Hping supports TCP, UDP, ICMP, and RAW-IP protocols. It is a network tool, assembler, and analyzer orienting and customizing TCP/IP packets. Moreover, Hping can trace route, send files between a covered channel, and many other features. Hping3 is an updated version of Hping [San06] [Too19].

3. **Nping**: is response analysis and response time measurement. It is a network tool that generates IP packets: header or raw packet. Nping virtually tunes any field of the protocol headers. Besides, Nping is a simple ping utility able to detect active hosts. Nping performs stack stress tests, ARP poisoning, DoS attacks, route tracing, and other purposes. [Lyo14].

4. **CICFlowMeter**: is a network tool that generates bidirectional flows and is able to extract more than 80 statistical network traffic features [CIC].

5. **Tensorflow**: is an open-source software for machine learning. It allows the creation of large-scale neural networks with many layers [SF18].

6. **Keras**: is a python library that helps to create deep learning models [Ket17].

7. **Pandas**: is a python library that uses for open-source, data analysis, and manipulation tools [M$^+$11].

5.2 Evaluation of the Inspector Device

Two works introduce the *Inspector* device, where section 4.2 explains how the work is completed step by step. So, this section shows the investigation work and simulation result In two subsections.

5.2.1 Feasibility of the Inspector Device

The work in subsection 4.2.1 compared to the PacketChecker module [DGLG17] to investigate the feasibility and the effect of existing the *Inspector* in the SDN controller. PacketChecker module deals with a flow of packets having the same forged addresses (e.g., 100 packets). But, in this experiment, every single packet has a different address. To do that, the Ryu controller manages a Fat-Tree topology [dSASZ17] with a server. Figure 5.1 illustrates the experimental topology and the Inspector. Each edge switch (from S1 to S8) connects up to 10 hosts. One of these 10 hosts acts as the attacker and periodically injects 10 forged packets (in an inter-packet interval of 0.01s) with different MAC and IP addresses based on Scapy [Mon18]. The network hosts send 76,000 request packets (ICMP) to the server in the experiment, including 40,000 forged packets.

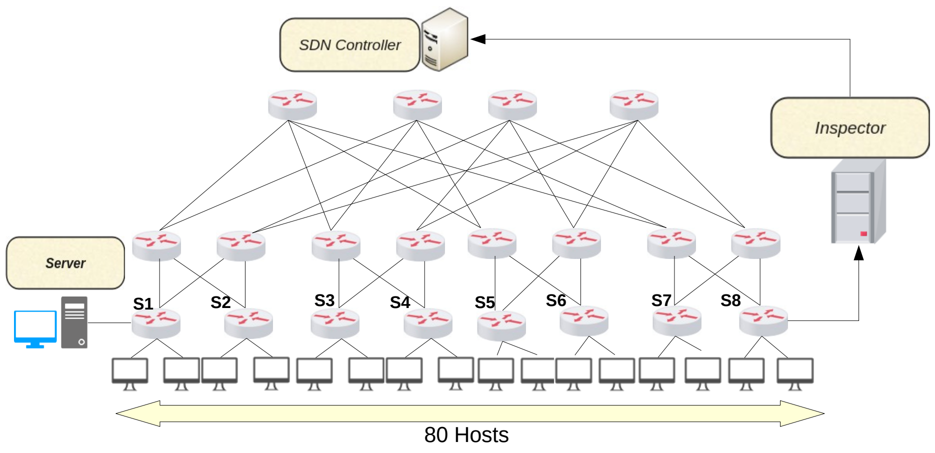


Figure 5.1. – The Experimental Topology

The implementation uses the Mininet emulator [Min21] to examine the Inspector's performance in a virtual machine working on Ubuntu 14.04 that has one process, an Intel Core i5-8400 2.80 GHz CPU and 12 GB memory. The experiment measures the number of TCAM entries, CPU usage, memory usage, and OpenFlow channel as following:

1. **TCAM**: The controller installs flow rule as *Packet_Out* Messages to the requesting switch as a response to the switch's *Packet_In* Messages. The implementation counts the flow table entries installed in the TCAM during the experiment. Therefore, the experiment uses CLI to instruct all hosts to send PingAll before the attack starts. Consequently, the OpenFlow switches have the completed flow rules that our topology needs to contact each other. Figure 5.2 manifests the number of created flow rules compared to that of the PacketChecker module in all the edge switches of the topology. That shows the *Inspector* can keep the TCAM's usage at the same level in case of no attack. Obviously, the improvement rate is between 200% and 400% and increases according to the number of attack attempts.

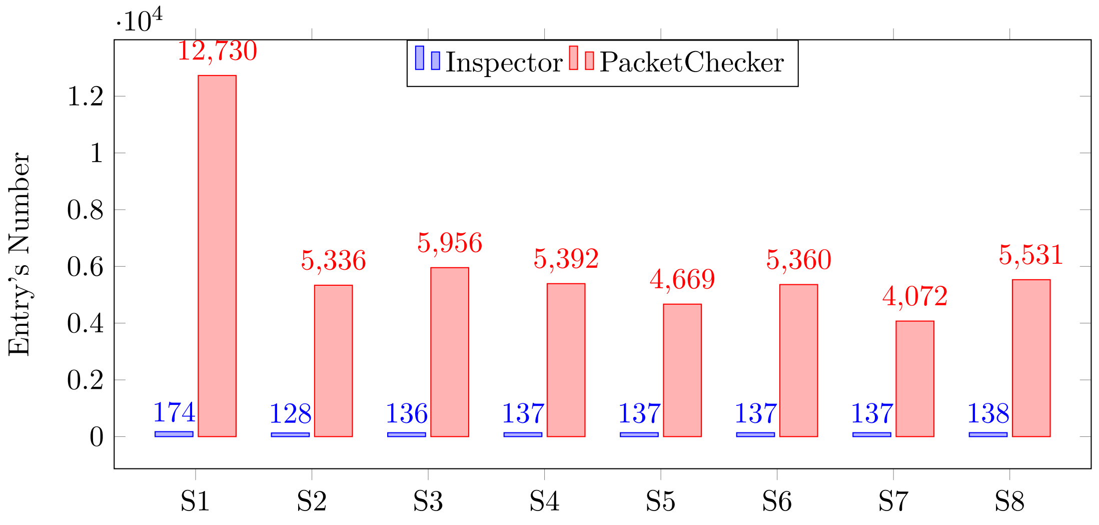


Figure 5.2. – TCAM Usage

2. **OpenFlow Channel Workload**: The OpenFlow channel is a secure channel to exchange transmitting messages between the Ryu controller and the OpenFlow switches [Oped]. The experiment measures the packets' number use the OpenFlow channel during the attack. Therefore, Figure 5.3 depicts the difference between the two models. Each point in the figure lines is the packets' number per 10 seconds interval.

   The *Inspector* can handle all the sent packets within 50% of the time used in the PacketChecker module. In this experiment, the *Inspector* spends 340 seconds while the PacketCkecker needs 670 seconds. This is because the PacketChecker module launches a procedure to deal with the incoming malicious packets. On the contrary, the *Inspector* drops them before they reach the controller.

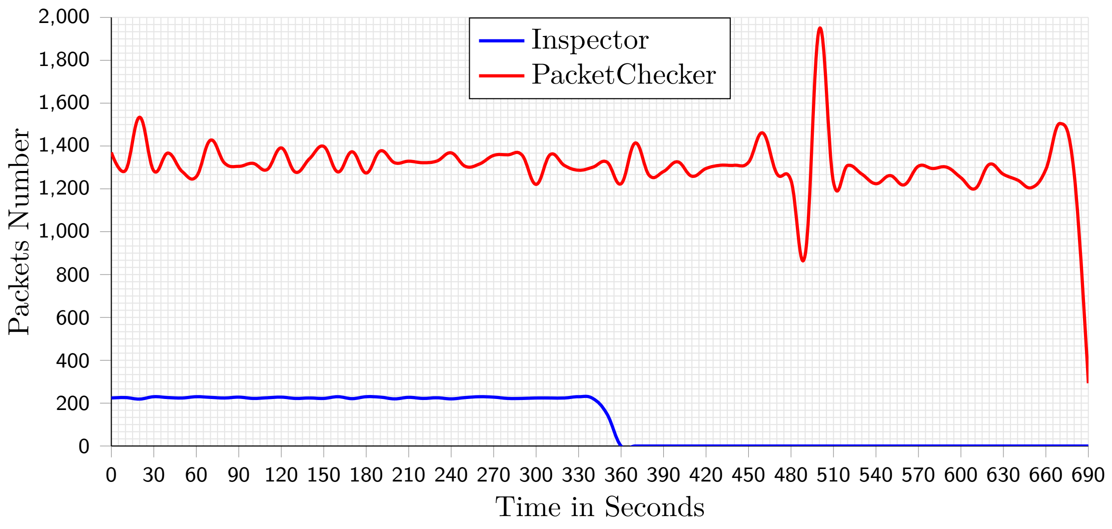


Figure 5.3. – OpenFlow Channel Workload

By comparing the number of dropped spoofing packets, the results show the *Inspector* could deal with almost all spoofing packets. The experiment transmits 76,000 packets, including 40,000 spoofing packets, and the Ryu controller received 36,000 packets (the legal packets only). Figure 5.3 presents the difference in the presence of the attack, which reaches around 85%. Efficiently, the *Inspector* neutralizes the *Packet_In* Messages and hence mitigates the load on the OpenFlow channel.

3. **CPU and Memory Usage**: The experiment evaluates the CPU usage and memory usage of the Ryu controller under attack using Psutil library [Psu].

   The implementation starts with a PingAll instruction to install the necessary flow rules into the switches. Then, the experiment launches the attack. Figure 5.4 presents a comparison of CPU usage between the *Inspector* and the PacketChecker module during the attack duration only.

   The *Inspector* minimizes the CPU usage to approximately null or 2% at the worst case because the *Inspector* denies the forged packets entering the controller. Still, the PacketChecker needs to deal with all the forged packets.

   Figure 5.5 depicts the memory usage during the attack duration only. In general, the PacketChecker module processes the topology management information from the attacker for the first time. After that, it denies other attempts with the same information. On the contrary, the *Inspector* does not allow any forged packet to enter the topology management service, hence the memory usage stays constant.

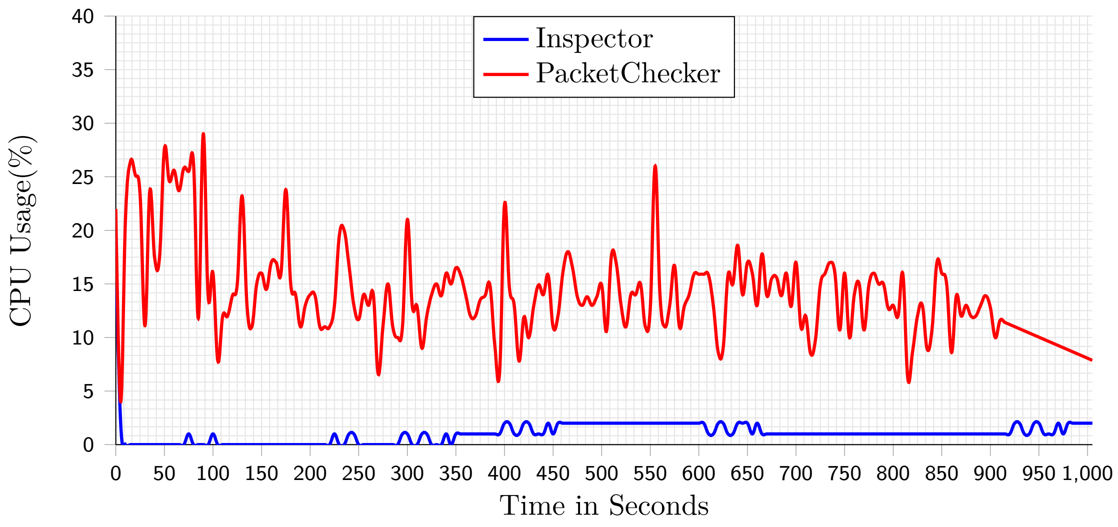


Figure 5.4. – CPU Usage

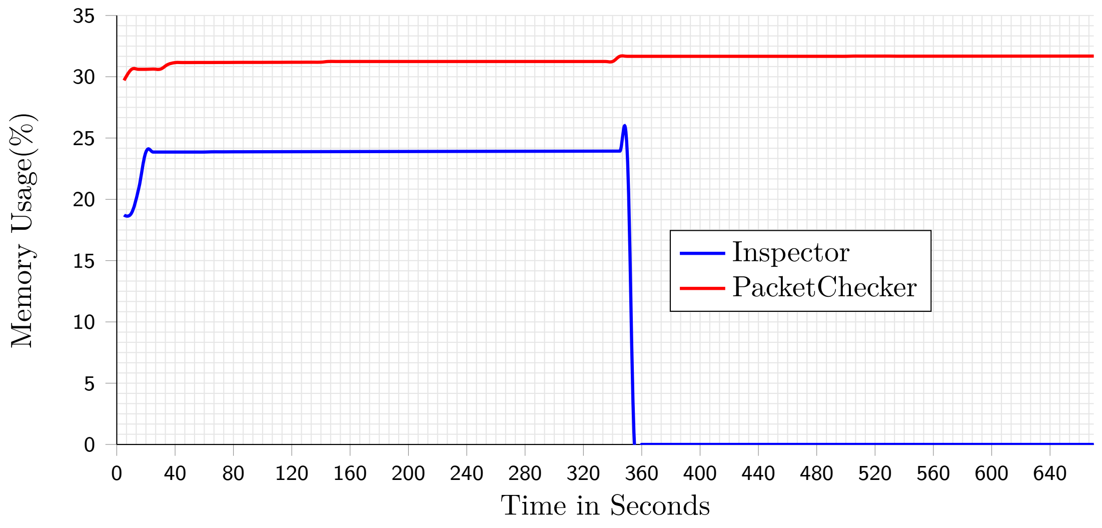


Figure 5.5. – Memory Usage

In the experiment, the *Inspector* decreases the memory usage between 22% and 38% compared to the PacketChecker.

4. **Delay Time**: According to carrier-grade recovery requirements in [SSC$^+$13], the time to send a packet from a source to a destination has to be less than 50 ms. That means the network performs in normal conditions to ensure that the *Inspector* does not influence the network performance. After the Pingall instruction, the network hosts sent 76,000 request packets to the server without attack. As a result, the round trip time's average for all sent packets has been around 13 ms, less than

50 ms. That means the *Inspector* does not influence the network performance badly.

5.2.2 Evaluating the Isolation of a Malicious Inspector

Subsection 4.2.3 presents the possibility to exploit the *Inspector* as a vulnerable point. This subsection shows the experiments regarding the effectiveness of the *Inspector* device against the DoS attack. The experiment uses the Mininet emulator [Min21] running on Ubuntu 18.04, an Intel Core i5-8400 2.80 GHz CPU, and 16 GB memory. In addition, the simulation implements the topology in Figure 5.1, where the Ryu controller 1.3 [Nip21] manages 20 OpenFlow switches in a Fat-Tree topology with a simple HTTP Server.

Therefore, the experiment assumes that the attacker controls the *Inspector* device to provide one malicious host an excellent chance to launch the attack towards the server. The topology has 80 hosts that respectively send TCP SYN packets to obtain the service from the server. Meanwhile, the attacker uses the NPING tool to generate a TCP SYN attack between 20 and 25 seconds. During the attack time, the attacker sends flow traffic with a high rate reach 3,000 packets/second [Lyo14]. This subsection shows the result of the experiments in terms of the workload of the controller resources, server bandwidth, and server CPU usage as the following:

1. Workload Overhead Assessment

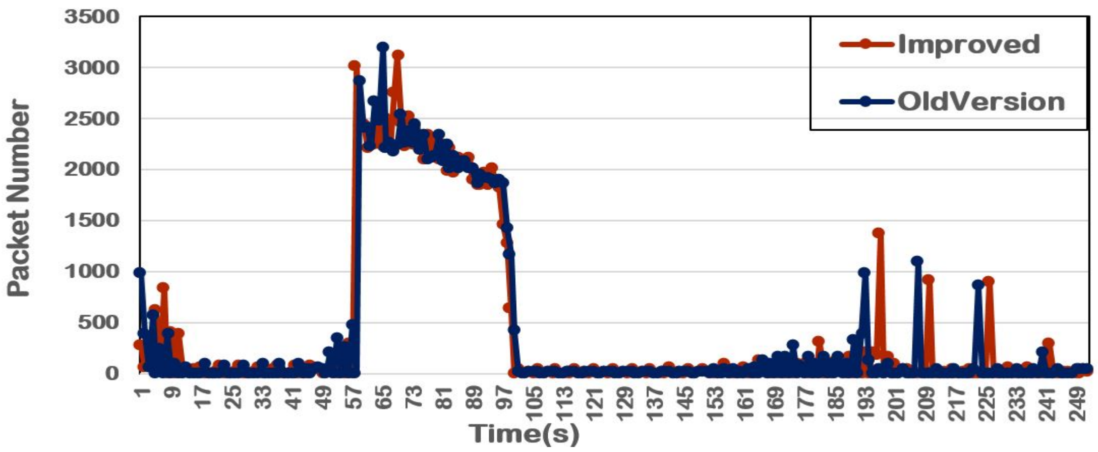


Figure 5.6. – The OpenFlow Channel Workload

To prove that the additional components do not cause a higher workload on the SDN controller, the experiment compares the previous *Inspector* approach with the improved version in terms of CPU usage, memory usage, and the workload of the secure channel. The Ryu controller collects traffic information every five seconds, where the controller receives the statistics of the edge switches. The implementation launches ICMP packets (PingAll) to compare the old version to the upgraded version in normal conditions. Then the hosts send TCP SYN packets to the web server. Figure 5.6 demonstrates the packets' number that hosts sent through the OpenFlow channel during the simulation. The behavior of the network is not much different because the amount of the statistics data is small and hence does not affect the secure channel.

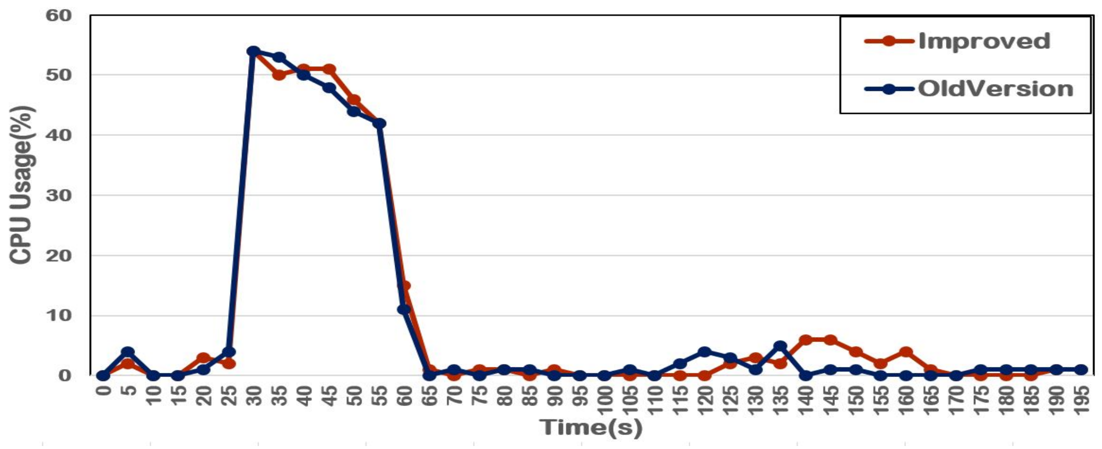


Figure 5.7. – CPU Usage of the Controller

Likewise, Figure 5.7 also shows the CPU usage for the two versions, and the Figure 5.8 presents the memory usage as well. Both figures show identical results in the different versions (difference 0.30%). The implementation shows that there is almost no noticeable change in CPU and memory usage.

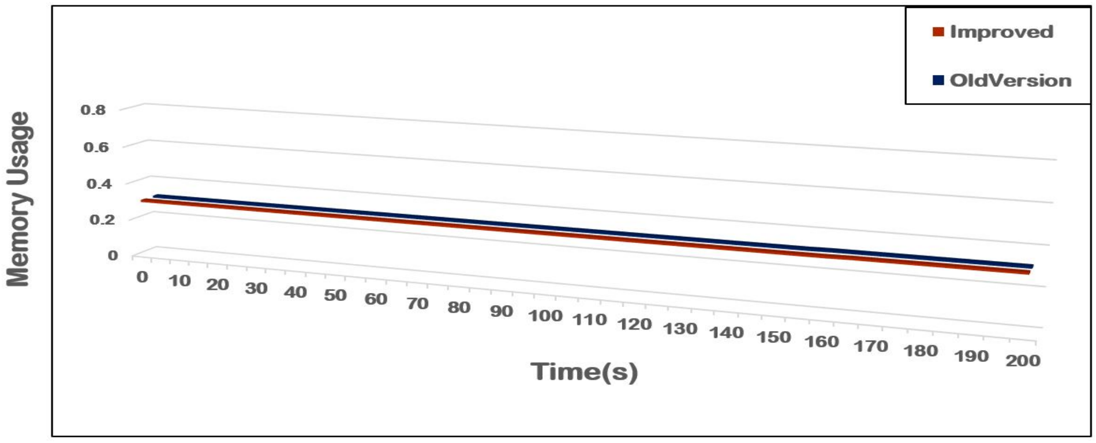


Figure 5.8. – Memory Usage of the Controller

2. Server Bandwidth

   After the experiment showed that the improvements do not lead to a high load on the Inspector, the simulation observed the attack web server. Therefore, we monitor the communication link utilization and the CPU usage. The experiment scenario introduces the attack's impact when the *Inspector* allows one malicious user to use the network. Figure 5.9 displays a comparison of the amount of data sent over open connections to the web server in the two versions. Thereby, the results appear the difference and demonstrate the need to enhance the Inspector's performance.

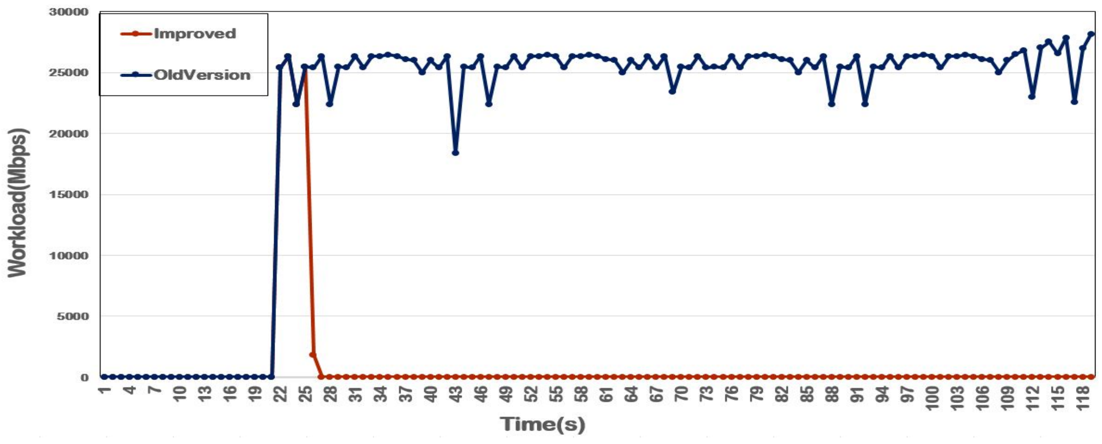


Figure 5.9. – Workload on the web server (MegaBits Per Second (Mbps))

   Once the Ryu controller detects that a particular port of the edge switch exceeds the threshold, the controller triggers the countermeasures. Therefore, the improved version mitigates the effect of the DDoS attack and keeps the server in the same conditions as before the DDoS attack. On the contrary, the previous version does not consider that the hacker controls the *Inspector* and then makes the possibility to attack the OSI application layer. Consequently, the attacker manages to make the channel between the server and the switch very busy.

3. Server CPU Usage

   The implementation uses the Psutil library to measure the CPU usage of the server by monitoring its process ID [Psu]. So, Figure 5.10 clarifies the attack's impact on the web server in terms of CPU usage. The simulation results demonstrate that the improved version can effectively block the DDoS attacks and allow the web server to sustain its regular operation. In contrast, CPU usage is always high

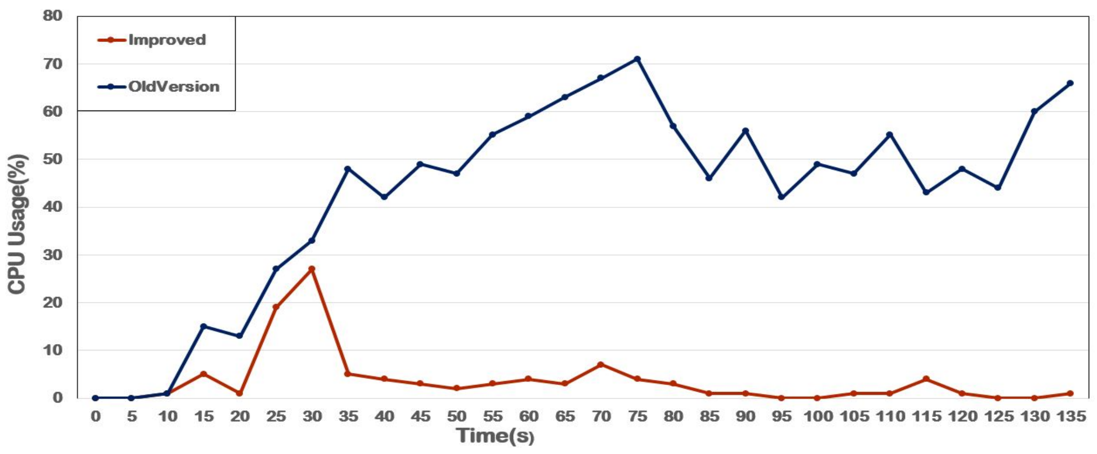


Figure 5.10. – CPU Usage of the Server

in the old version during the attack. The improved version can keep the server resources in satisfactory condition.

5.3 The Performance Investigation of the Convolutional Neural Network Component

Subsection 4.3 exhibits the second component that relies on entropy equation to monitor the traffic of the OpenFlow channel. Moreover, the second component depends on the CICFlowMeter tool to extract the traffic feature. In a suspicious behavior, the 1D-CNN model uses the extracted features as an input to ensure the suspicious traffic.

This section presents the details of the experimental setup to analyze the performance of the proposed method. Therefore, the experiments are separated into two subsections. The subsection 5.3.2 discusses the experimental results of the proposed model (1D-CNN). In addition, the subsection 5.3.3 discusses the model's implementation on the SDN environment. But, the next subsection 5.3.1 firstly explains the used dataset to train and test the 1D-CNN model.

5.3.1 CICDDoS2019 Dataset

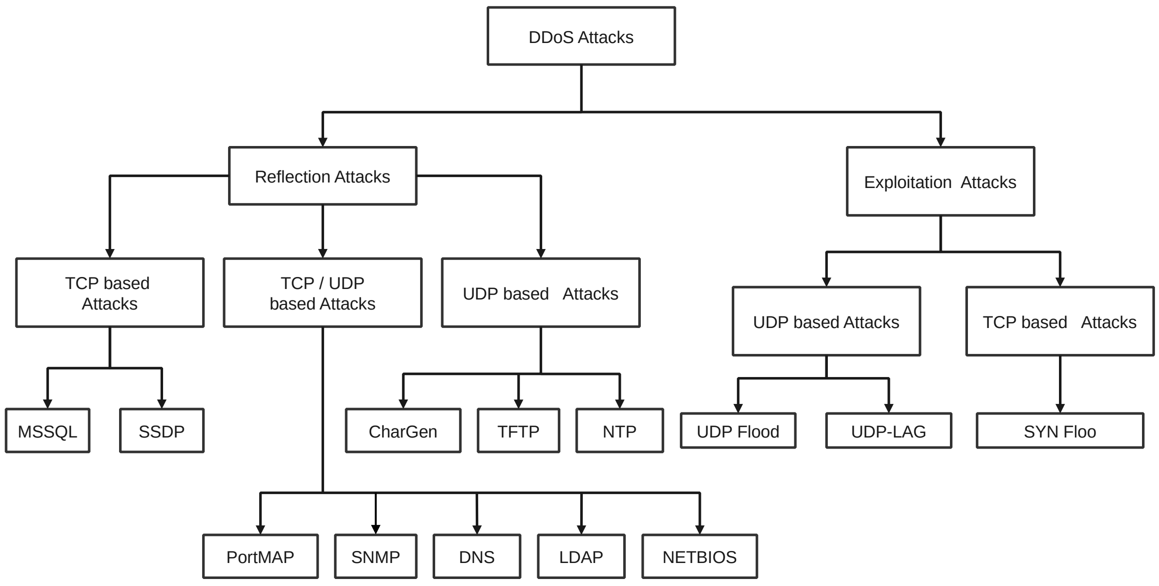


Figure 5.11. – The Taxonomy of DDoS Attacks in CICDDoS2019

CICDDoS2019 is one of the most significant modern DDoS datasets and includes many attacks types and benign traffic [fC19]. The dataset is available in both CSV and PCAP formats. The researchers used the CICFlowMeter tool [CIC] to generate the CSV files that contain the traffics along with more than 80 extracted features (see Appendix C). The PCAP file is the real-world captured date and helps to resend the traffic if needed. The researchers used real machines to generate the CICDDoS2019 dataset in two days. The first day was dedicated for the training and contained 12 types of DDoS attacks as follows: SYN, WebDDoS, MSSQL, UDP, SNMP, NetBIOS, DNS, LDAP, TFTP, NTP, UDP-Lag, and SSDP DDoS based attack. On the second day, the testing data ware collected and included seven DDoS attacks: SYN, MSSQL, LDAP, UDP, Port Scan, UDP-Lag, and NetBIOS. In addition, they used 25 users to simulate the benign traffic based on HTTP, HTTPS, File Transfer Protocol (FTP), Secure Shell (SSH), and email protocols [fC19] [SLHG19].

Figure 5.11 shows a full list for all available DDoS attacks in CICDDoS2019 and introduces a new DDoS taxonomy dividing the DDoS attacks into two groups [SLHG19]:

1. **Reflection-based DDoS**: which means that the attacker's identity stays hidden and the attacker exploits a legitimate third-party component (subsection 3.2.1 explains the method). As Figure 5.11 shows under reflection attacks category,

   Firstly, the TCP based attacks include:

   - Microsoft SQL Server (MSSQL): The attacker exploits an SQL injection vulnerability to go around the application security and authenticate as the administrator.
   - A Simple Service Discovery Protocol (SSDP): The attacker exploits Universal Plug and Play (UPnP) networking protocols to direct a volume amount of traffic to the targeted victim. The attacker spoofs the victim's address and requests the server. So, the server sends the response to the victim.

   Secondly, UDP based attacks include:

   - CharGen: The attacker submits small packets (queries) carrying a crafted IP of the target to internet-enabled devices running CharGEN. Then, the UDP server uses the crafted queries to send UDP floods as responses from these devices to the target.
   - NTP: The attacker exploits publically accessible NTP servers to overwhelm the targeted by UDP traffic.
   - Trivial File Transfer Protocol (TFTP): the attacker sends a request using the victim address. Therefore, the server responses with big size files than the original request.

   While the attacker can carry out some attacks using either TCP or UDP like:

   - Domine Name System (DNS): DNS server is responsible for transferring the website's name into IP addresses. The attacker sends a huge number of requests to the DNS server to overwhelm it.
   - Lightweight Directory Access Protocol (LDAP): When an application fails to sanitize user input properly, the attacker modifies LDAP statements through techniques similar to SQL Injection.

- NETBIOS: The attacker views or accesses (delete, copy, modify, ..etc.) any shared file on an accessible computer by utilizing the NetBIOS service port139.
- Simple Network Management Protocol (SNMP): The attacker sends enormous volumes of requests, which can be directed at victim targets from multiple networks.

2. **Exploitation-based Attacks**: The attacker controls applications, networks, operating systems, or hardware through their vulnerabilities advantages. Then, the attacker sends a vast amount of packets towards the victim. According to the Figure 5.11, TCP-based exploitation attacks include SYN flood, and UDP-based attacks include UDP flood and UDP-Lag.

   In UDP flood attack, the attacker sends a large volume of UDP packets to the victim at a very high rate. In TCP SYN attack, the attacker consumes server resources by exploiting three-way handshake. The attacker sends repeated SYN packets to the target machine until the server crashes/malfunctions. In the UDP-Lag attack, the attacker disrupts the connection between the client and the server. The attacker primarily uses the UDP-Lag attack in online gaming. Therefore, the attacker uses a hardware switch known as a lag switch or a software program that runs on the network and hogs the bandwidth of other users.

5.3.2 Evaluation Criteria for One Dimension Convolutional Neural Network

The experiments use four metrics to evaluate the 1D-CNN model, namely Accuracy, Precision, Recall, and F1 score. Consequently, the experiment has a systematic benchmarking analysis with other related approaches. The mathematical representation of these indicators are clarified based on the following equation:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

$$F1\ Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{5.4}$$

True Positive (TP) and True Negative (TN) represent the values that are correctly predicted. In contrast, False Positives (FP) and False Negatives (FN) are wrongly classified events. All four metrics compare the experiments against the four machine learning models (ID3, random forest, Naïve Bayes, and logistic regression) proposed in [SLHG19] with CICDDoS2019 for performance validation. The experiments use the Python programming language to build all models by TensorFlow, and Keras framework [AAB+16, Ker]. Moreover, the experiments consider the best practice and trial to construct 1D-CNN with the best values for hyper-parameters [ELKJ20]. The experiments were performed on Ubuntu 20.04 LTS 64-bit operating system with Intel®Core™ i5-8400, CPU @ 2.80GHz × 6, 16 GB of RAM, and AMD® Rv635 Graphics.

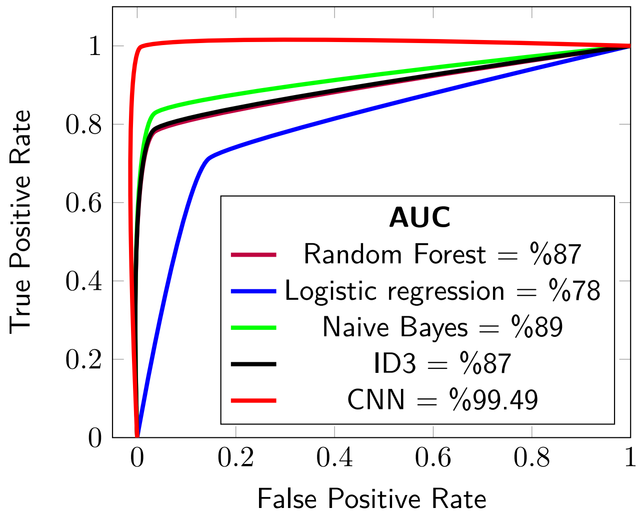


Figure 5.12. – Receiver Operating Characteristic (ROC)

Firstly, the Receiver Operating Characteristic (ROC) curve evaluates the model's ability to perform accurately. The ROC curve plots the relation between True and False classes. The Area Underneath the ROC Curve (AUC) measures the separability between false positive and true positive rates. Figure 5.12 depicts that the 1D-CNN model provides an AUC of 99.49%, which means the ability to separate 99.49% of positive and negative classes correctly. Moreover, the 1D-CNN model introduces a notable enhancement compared to other machine learning models. 1D-CNN provides 12% 22%, 10%, and 12% enhancement compared to Random Forest, Logistic regression, Naïve Bayes, and ID3, respectively.

Learning Model	*accuracy*	*precision*	*recall*	*F1 score*
Random Forest	0.90	0.92	0.78	0.84
Logistic Regression	0.80	0.73	0.72	0.72
Naïve Bayes	0.91	0.92	0.82	0.87
ID3	0.90	0.92	0.78	0.85
1D-CNN	0.9945	0.9884	0.9962	0.9923

Table 5.1. – Detection Performance Comparison

Table 5.1 shows more details about the superiority of the 1D-CNN model over the other models. The enhancements are presented in terms of accuracy, precision, recall, and F1 score. 1D-CNN achieves better results with around (10% - 20%) accuracy enhancements compared to the other models. Additionally, the 1D-CNN model shows the best results in terms of precision with (6% - 25%) enhancements. In terms of recall parameter with (17% - 27%), and (12% - 27%) in terms of F1 score. Table 5.2 illustrates the confusion matrix information to describe the classification performance of 1D-CNN. The table shows the correct and false predictions. The table proves that the 1D-CNN model outperforms all other machine learning models in four different events (TP, FP, TN, FN).

Learning Model	*TN*	*FP*	*FN*	*TP*
Random Forest	0.96	0.04	0.21	0.79
Logistic Regression	0.86	0.14	0.28	0.72
Naïve Bayes	0.96	0.04	0.17	0.83
ID3	0.97	0.03	0.21	0.79
1D-CNN	0.99	0.006	0.003	0.99

Table 5.2. – Confusion Matrix

5.3.3 Effectiveness of One Dimension Convolutional Neural Network In SDN

The experiment uses the Mininet emulator to investigate the performance of the used 1D-CNN model based on entropy information. Also, the investigation evaluates a small network with two switches (S1, S2) managed by Ryu Controller [Nip21]. S1 connects to four normal users and four attackers. Also, S2 connects to four servers. The normal users use Scapy [Sca] to read benign UDP PCAP files, which have numbers from 500 to 749 in the CICDDoS2019 dataset, and send these packets to the servers. After a short time, the attackers also start reading the malicious UDP PCAP files, which have numbers from 750 to 818 in CICDDoS2019, spoofing the MAC address for every packet. The experiment executes the simulation twice to check out the entropy behavior for crafted packets with MAC source or destination in both cases.

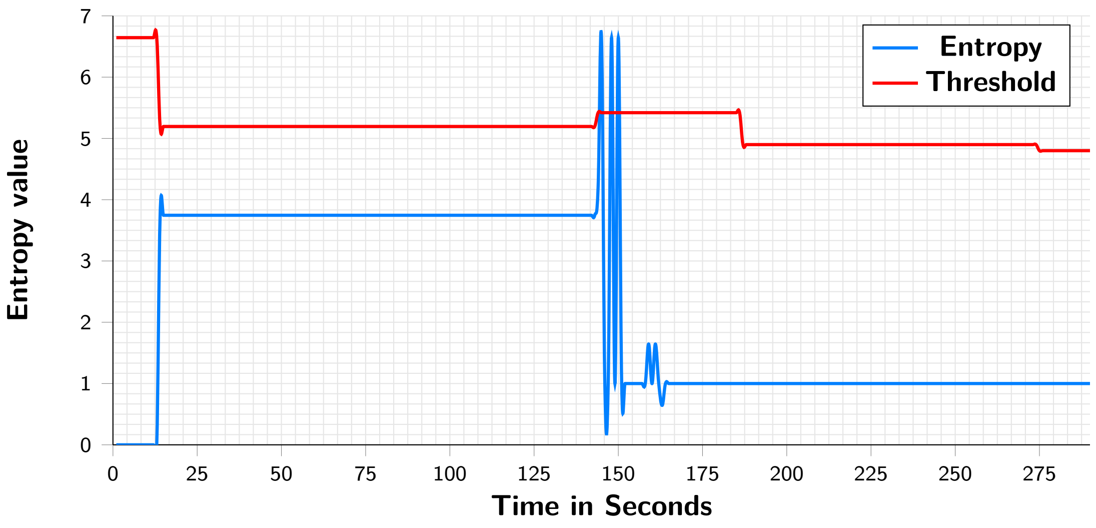


Figure 5.13. – Destination Entropy Value

Figures 5.13 and 5.14 depict the behavior of the entropy values during the simulation time. the figures show the entropy behavior when the attacker crafts MAC source address or destination source address. In the beginning, the threshold value started from the highest value for log_2 of the window size(100), which is equal to 6.643. Once the packets start entering the network, the threshold will change because of the new *Packet_In* messages. The simulation monitors the last entropy value. The entropy value will not change when the users' communications do not need to request the controller for intervention. That is clear from 19 until the beginning of the attack, around 140. The controller stops the attack until the end. The entropy value crosses the threshold more than once in both figures because the attacker started the attack after reading the PCAP files, launching the attack at different times. However, the features could be different, where the switch table identifies the packet according to the used SDN controller instructions. For example, IP addresses, MAC addresses, protocol, output or input port, VLAN, MPLS, etc. [XWX20].

Figure 5.15 clarifies the behavior of the CPU usage in three states: (1) with 1D-CNN controller structure based on entropy information, (2) normal without DDoS attack and (3) attack state without defense mechanism. In the normal state, the CPU usage rate is high between 0 and 25. The reason that normal traffic is sent through the network for

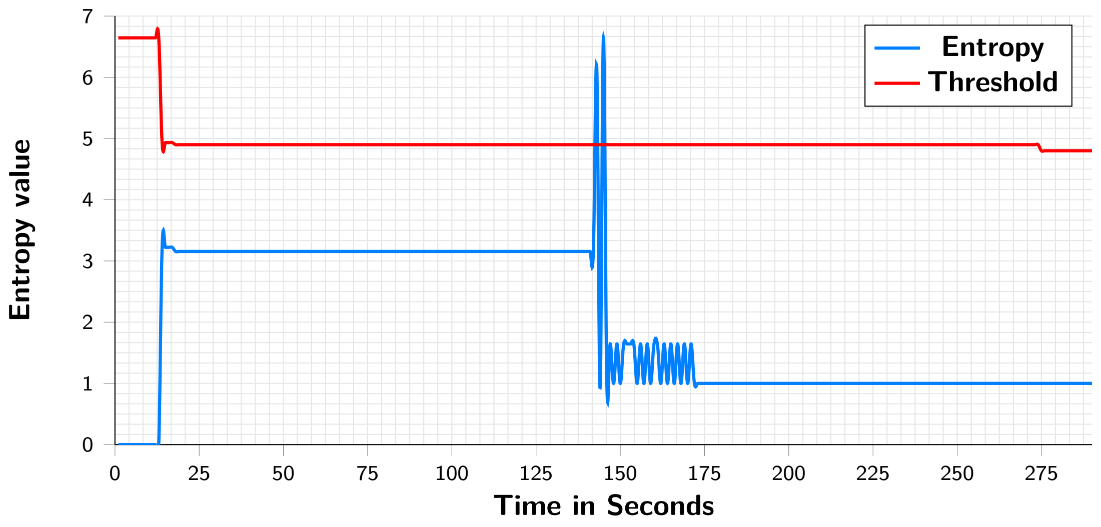


Figure 5.14. – Source Entropy Value

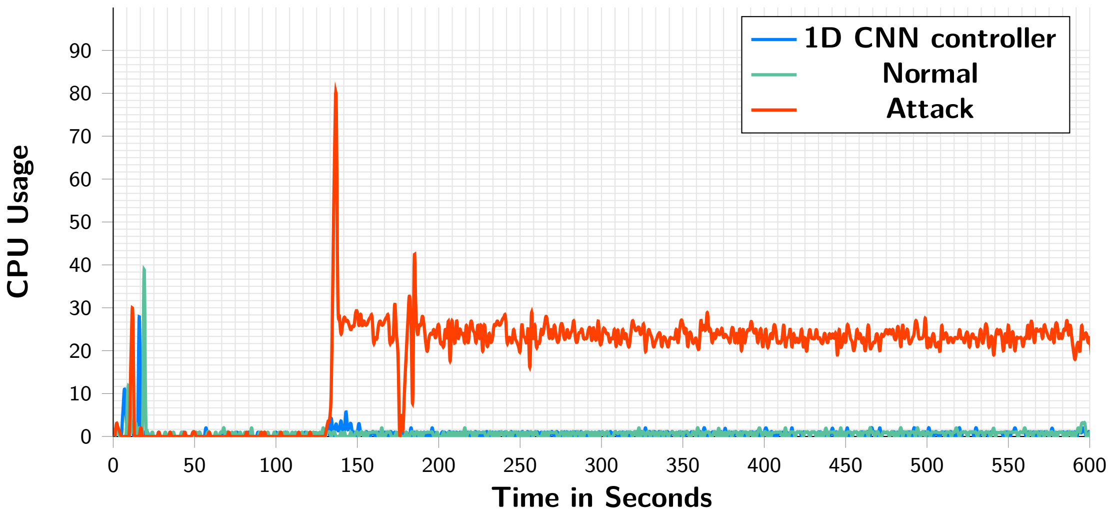


Figure 5.15. – CPU Usage

the first time, and the OpenFlow switch triggered a relatively large number of *Packet_In* messages to establish flow entries. In the attack state, a very high rate of CPU usage is notable when the simulation launched the malicious traffic (after around 135 seconds) in addition to the high rate in the beginning. However, this does not occur with our proposed method. The CPU usage is high at the start only, increasing slightly before the controller blocks the connected ports. The controller needs a short time to read the last 100 traffic flows saved as a CSV file to verify any attack towards the controller during the past time.

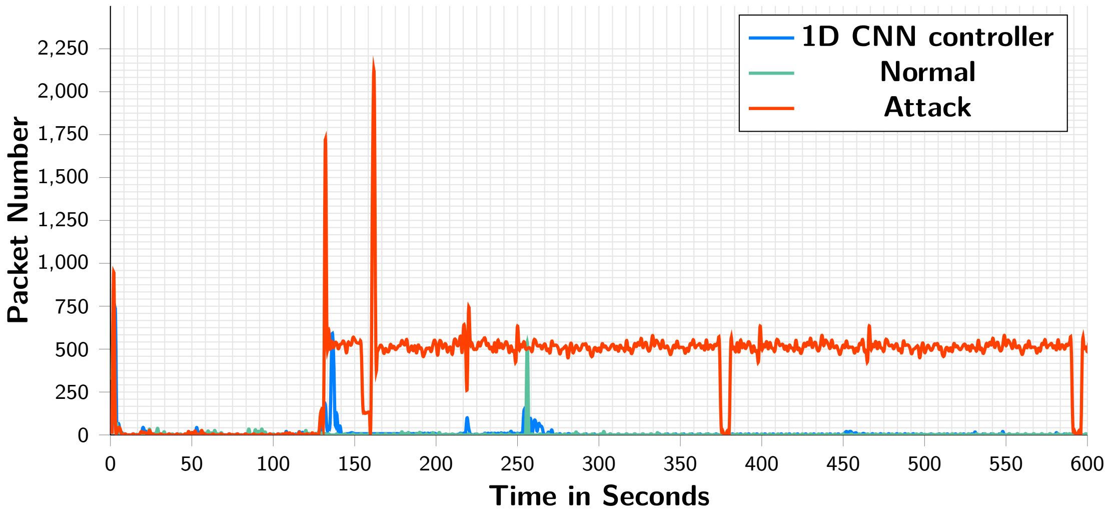


Figure 5.16. – OpenFlow Channel Bandwidth

Figure 5.16 introduces the amount of *Packet_In* messages through the OpenFlow channel during the simulation time. In the presence of the attack, the channel is crowded. But, with the second component, the channel returns to behave in a normal state quickly. Finally, the controller installs instructions by *Packet_Out* messages to respond to the switch's *Packet_In* message.

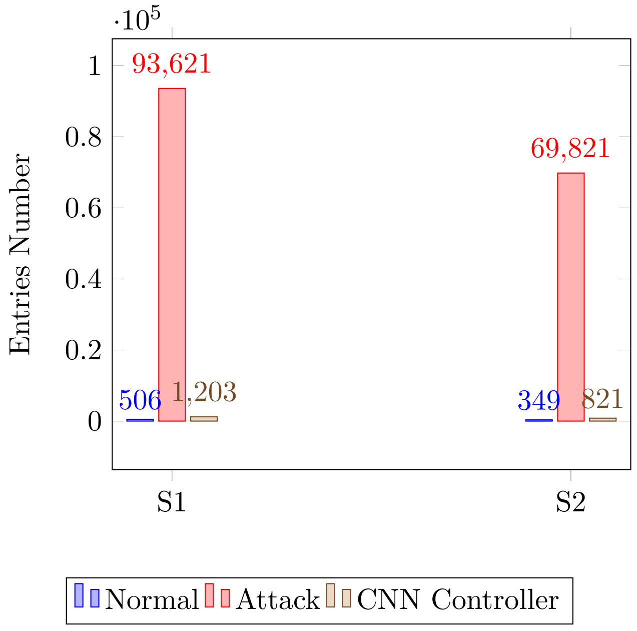


Figure 5.17. – Number of Installed Flow Entries

Figure 5.17 measures the number of the flow table entries installed in the flow table during the experiment. To count the number of entries that the topology needs, users sent PingAll packets before the attack started. As a result, the number of installed entries is enormous in the absence of the defense mechanism, making the switch unable to forward the packets because of the lack of resources. On the contrary, the second component prevents the attack. It minimizes the effect, which is evident with the installed entries in both normal states and our proposed method.

5.4 Performance Evaluation for Applying IPSec Concept

Section 4.4 clarifies how the IPSec concept could be helpful to distinguish between the attacker and benign user. Moreover, how the suggested statistical methods perform to detect the attack on the SDN controller or the Infrastructure layer.

This section shows details about the implementation results and the simulations scenarios. Then, the section analyzes the effectiveness of the implemented attacks and the performance of the proposed countermeasures. Therefore, the experiments are applied on ubuntu 18.04.5 LTS with 6 core Intel Core i5-8400 and 16 GB RAM size. Besides, the experiments use the Mininet emulator to implement all the target scenarios. Mininet provides a realistic virtual environment to create the network scenario, and the Ryu controller as the SDN controller [Min21, Nip21]. Furthermore, Python configures the users with the requested algorithms, and Scapy is the primary tool responsible for crafting and generating the packets [Sca].

Figure 4.9 (see subsection 4.4.3) describes the implemented topology consisting of 11 OpenFlow switches and one Ryu controller. The simulation also uses different numbers of users in several scenarios. This way, the experiment tries to explain the effect of the proposed countermeasures on the different types of protocols (TCP, ICMP, and UDP) and various network environments (Ethernet, wireless). The link bandwidth is 100 Mbps for the connected ports in the edge switches and 1 Gbps for all core links between switches. Eventually, the link delay is 5 ms, and 100 is the max buffer queue size for all non-edge ports.

5.4.1 Effectiveness of the Proposed Method Against UDP Flood attack

UDP flooding is a DoS attack, where the attacker sends numerous UDP packets toward a targeted receiver (user, server). Thereby, the attack overloads the victim's resources, which is limited, to destroy them such as forwarding nodes and the link's bandwidth.

Moreover, the UDP flooding primarily exploits the action taken by the receiver to respond to a UDP packet sent to one of its ports. Once the user receives a UDP packet at a particular port, it responds with two proceedings, either with the requested service or ICMP packet, to inform the sender that the destination is unreachable. That means more work on the network resources because each malicious UDP packet generates an ICMP packet.

The experiment simulates the Proposed method to evaluate the impact of UDP flooding in a normal situation in terms of the delivery ratio, bandwidth usage, and packet delay. Furthermore, the implemented scenario configures both the sender and servers. 12 hosts are distributed on 4 edge switches, where each switch has three users: client (benign user), server, and attacker. The benign user randomly sends one of the servers not connected with the same edge between (5-35) of size 512 bytes packet per second. But, the attackers use the $Hping3$ tool to transmit huge volume packets (Size = 1024 bytes) to a server that is not connected with the same edge switch to spread the packets through the network.

1. **Packet delivery ratio**: Packet delivery ratio is the rate of the packets' number that the server received compared to that the client sent. The core switches drop the packet because of a buffer overflow caused by the high-speed attacker traffic.

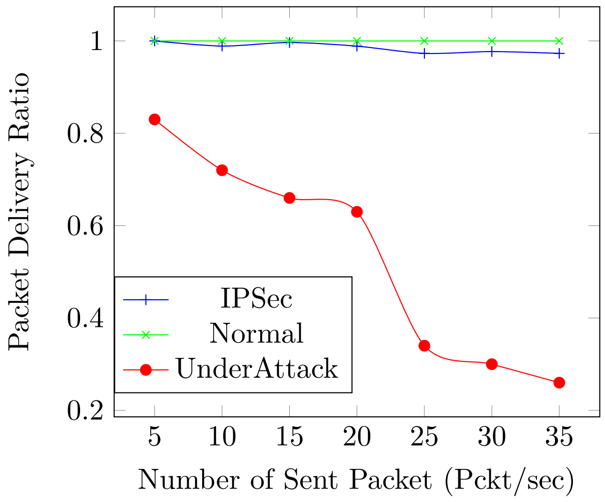


Figure 5.18. – Packet Delivery Ratio

In Figure 5.18, the attack's effect is evident on the packet delivery ratio. When the normal packets increase, the chance to lose more packets is higher. So, if the benign packets increase under attack conditions, the packet delivery ratio decreases. The switches drop more packets because the malicious traffic overloads the switch buffers.

In the presence of the IPSec countermeasure, the controller rapidly detects malicious users, removes all exploited flow entries, and blocks non singed packets. As a result, the behavior of the packet delivery ratio under the IPSec controller management is very close to the normal situation.

2. **Bandwidth usage** : Bandwidth usage is the impact ratio of malicious traffic on the bandwidth consumption at UDP server links. Figure 5.19 introduces the average of the bandwidth usage in KBps. If there is an attack, the traffic increases because of increasing the sent packets. Also, the packet number would not lead to congestion over the connected links and the OpenFlow Switches buffers. The bandwidth usage is around 50% of the link bandwidth (100 Mbps). Thus, it causes more congestion and adds long delays for sent packets.

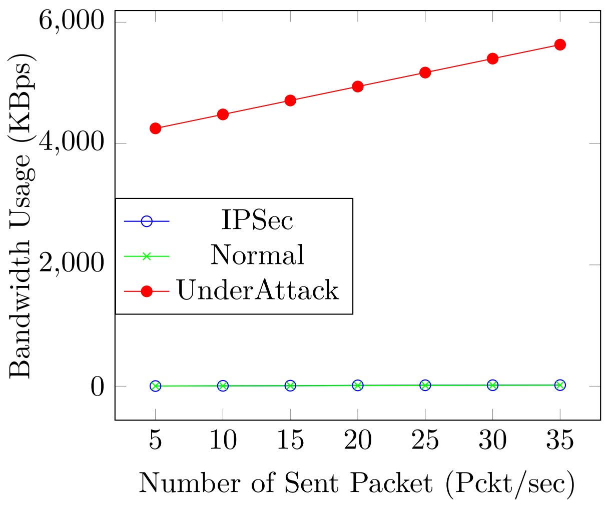


Figure 5.19. – Bandwidth Rate

3. **Delay Time**: In Figure 5.20, the delay has the same conduct as the previous metrics. The delay increased when the sent packets increase because of the limited queue buffers or link congestion. In addition, the attacker crafts the malicious packet with proper headers, which means the exploited flow entries transferred all packets. However, the IPSec controller blocks the malicious packets, and their influence would disappear from the network. Finally, additional delays in data delivery would also be disappeared by blocking the malicious traffics.

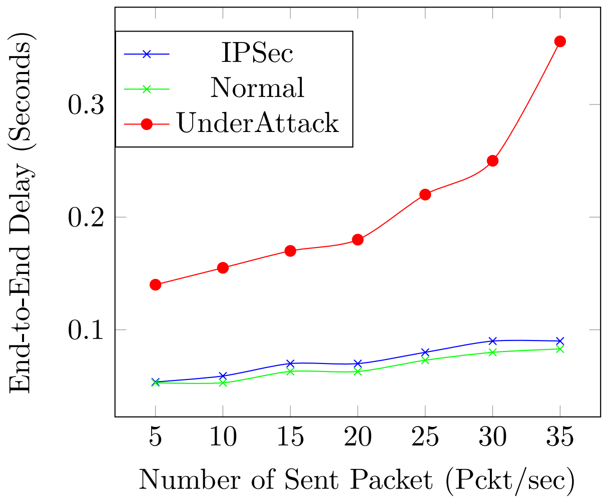


Figure 5.20. – End to End Delay

5.4.2 Effectiveness of the Proposed Method Against SYN Flooding Attack

SYN flood attack overwhelms the target to make it unavailable to benign users traffic. So, the attacker aims to locate all available server resources with a half-open connection. Generally, the TCP connection needs to determine whether the receiving host is ready to receive the data or not. Therefore, TCP user uses a three-way handshake. TCP starts the process when the user sends a request packet that has the SYN flag. In response, the server replies with the SYN-ACK. Finally, the TCP user sends an ACK back to the server and starts sending or receiving the data.

When the attacker malignantly sends tremendous SYN packets to locate all available ports on the victim's machine. Consequently, the victim would heavily answer the legitimate traffic or cannot answer at all. The evaluation compares the IPSec controller to the entropy-based detection (EBD) technique [CLMR19]. EBD Suggests a solution to count micro traffic in a modified edge switch. If the number exceeds a fixed number, the edge switch alarms the controller to block the malicious Inport.

Moreover, the evaluation compares the IPSec controller to SLICOTS, which counts the uncompleted TCP connection and then blocks the malicious Inport, which exceeds a fixed number (10 - 100). The target scenario is similar to the EBD and the SLICOTS paper [CLMR19, MJC17]. So, the number of the attacker is 120, while the benign hosts are two and the topology is introduced in Figure 4.9. Each benign user sends one or two requests per second and each request sends between two-five packets.

Based on those mentioned earlier, the experiment evaluates the detection time in response speed (scalability) and the FPR (the percentage of malicious packets wrongly identified as a benign packet) or 1-specificity (is the percentage of malicious SYN packets which the OpenFlow switch incorrectly identified as benign during the attack time (sensitivity)). Thus, the attacker performs a dataset (The Center for Applied Internet Data Analysis (CAIDA) has released DDoS Attack 2007) thst contains the attack traffic to the victim nodes [Too07]. The Scapy tool reads the DDoS Attack 2007 Dataset. Then, it generates the DDoS flooding attack traffic from the attackers to the TCP server. The dataset includes around one hour of anonymized traffic traces from a DDoS attack (TCP, UDP, and ICMP). The traces include only attack traffic to the victim and responses from the victim. The researchers distributed the one-hour trace in 5-minute pcap files, where the data's size is 5.3 GB (21 GB uncompressed). Also, they removed the payload from all packets.

The experiment configured the EBD technique with the same parameters used with the target scenario. 6 seconds as monitoring and M = 3 (M is the repeat of exceeding the threshold). The dedicated algorithm calculates the threshold entropy in work [CLMR19]. At the same time, for the SLICOTS, the fixed number for the uncompleted TCP connection is varied between 10 and 100. In this subsection, the experiments evaluate the ability of the Proposed method to counter the SYN flooding attack as the following:

1. **Detection time**: After 40 iterations of the experiments, the results of the detection time, in Figure 5.21, show that the IPSec controller has the best response time on average compared to the other proposal. The reason behind this result is that the attack sometimes starts in the middle of the round, so the controller does not need to wait for 3 seconds to note the attack and make a decision. In addition, the IPSec controller needs to directly compare the imported numbers to the predefined threshold before deleting the exploited entries without doing more prior tasks.

2. **1-specificity**: IPSec reaches almost 100% TPR (the percentage of benign packets correctly identified as benign packets) when it has approximately 0.087 FPR.

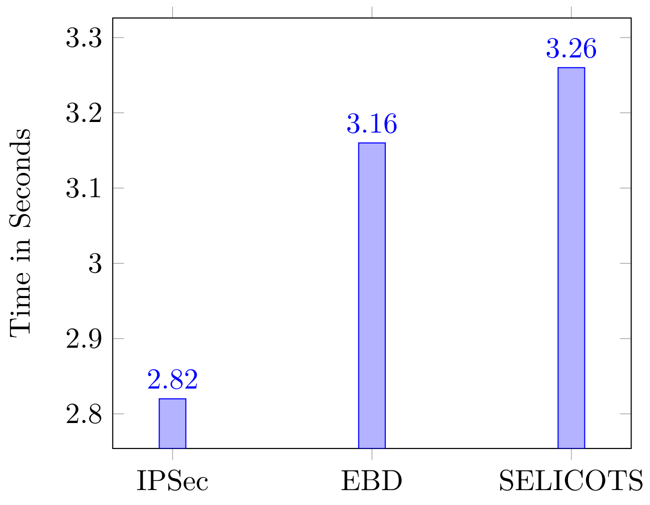


Figure 5.21. – Detection Time

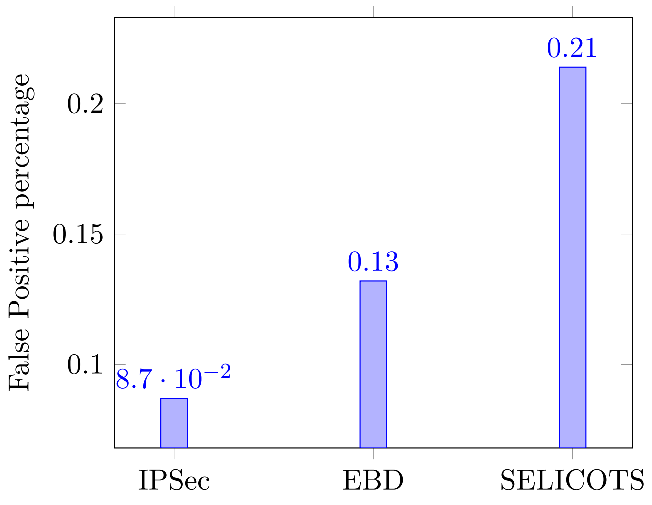


Figure 5.22. – False Positive Rate before the Countermeasure

Figure 5.22 depicts how much it is essential to detect the attack and stop it early. On the contrary, the EBD gets 100% TPR with 0.132 FPR while 0.214 for the SLICOTS results.Consequently, the FPR of the IPSec's proposal is lower than the EBD and SELICOTS FPR by 34% and 60% respectively.

3. **Bandwidth usage with freeloading attack**: The scenario has minor changes to verify the ability of the IPSec proposal to handle the crafted packet compared to the previous work. Four benign users are connected to edge switches by the wireless access point. They begin transmitting a TCP packet toward a TCP server. Each user can send 1 or 2 requests per second and between 2 - 10 data packets for the request randomly. After 60 seconds from launching the simulation, four

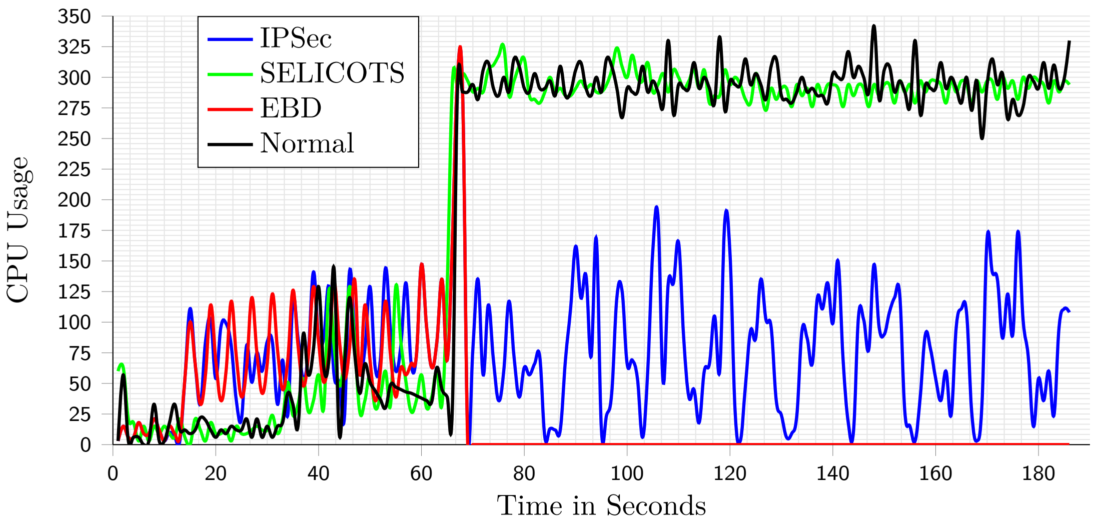


Figure 5.23. – IPsec's Effectiveness against the Freeloading Attack

attackers start using the dataset mentioned above (CAIDA 2007) to launch the TCP server's attack.

Figure 5.23 demonstrates the effect of this type on the TCP server where the attacker exploits the installed flow entries to attack the network resources. The bandwidth has been normal for the first time of the attack. But, when the attackers craft a massive volume of packets and send them to the victim node, the connected link bandwidth increases. Distinctly, it is noticeable that the EBD controller prevents the attack by blocking the Causative *In_Port*. Still, this way would block the benign user from reaching the network services. SELICOTS behaves like the standard controller because both of them have no way to detect the freeloading attack. So the increase in the link bandwidth is perspicuous enough to stop the TCP server and other forwarding resources in the data plane. Nevertheless, the IPSec proposal distinguishes between benign users and malicious users. Using AH is sufficient to make the switch enables to prevent the attack and foreword the benign user.

5.5 Evaluation of the Recurrent Neural Networks Models

The section 4.5 confers a suggestion to enhance the way to detect the DDoS attacks when the framework uses the IPSec concept as a countermeasure. The RNN models Depend on request the flow table from the OpenFlow switches to analyze the traffic behavior. This means the models are an enhanced solution to the third IDS component in section 4.4. This section uses the InSDN Dataset to evaluate the ability of the DLAs to detect. Therefore, the section introduces the used dataset and explains detection evaluation metrics. Afterward, the section describes the experimental setup. In the end, the suggested work is compared to another state-of-the-art approach.

5.5.1 InSDN Dataset

The InSDN dataset is one of the state of the art datasets for IDS evaluation in the context of SDNs. The dataset includes benign traffic and various attack categories that can occur in the different elements of the SDN platform [ELKJ20].

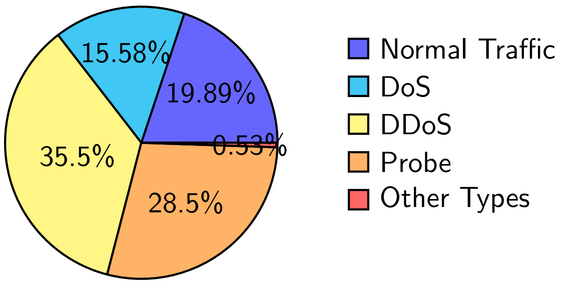


Figure 5.24. – InSDN Dataset Distribution

Figure 5.24 explains the dataset distribution that has different types of attacks and includes seven attack classes:

1. **DoS attack**: (1) DoS attack towards the SDN controller. (2) DoS attack to affect benign users, network bandwidth or any network device using protocols such as UDP, TCP or ICMP. (3) DoS attacks that overflow a server or the application layer on the victim device using the HTTP protocol.

2. **DDoS Attacks**: TCP-SYN Flood, UDP Flood, and ICMP Flood attacks from different sources towards the same victim in parallel.

3. **Probe**: The probing attack obtains information about the victim using scanning the operation system, discovering the open ports, etc.

4. **Botnet**: The attacker controls some benign users by malware and runs different malicious activities. For instance, they are stealing information, fraud attack, launching DDoS attacks against victim servers or web applications servers.

5. **Web Attack**: When the user accesses a website, the attacker installs a malicious code. As, the malicious code executes, the attacker obtains information from

the client machine, such as session tokens, cookies, etc. Moreover, the attacker could access the database of a server by injecting malicious SQL code into the web application server. Thereby, the attacker has the power to change, delete, misuse, and steal information stored in the database.

6. **Password Brute-Force Attack (BFA)**: The attacker creates a dictionary for all user names and password credentials and tries all of them.
7. **Exploitation User-to-Root (U2R)**: A normal user illegally accesses either root's or super user's privileges (such as admin) in the network.

5.5.2 Evaluation Metrics and Experimental Results

This experiment extends the previous approach with GRU to work with RNN and LSTM. Firstly, the suggested model selects 48 features (as the Appendix B clarifies) to be input for the used algorithms [ELKJ20]. Then, the experiment compares the result to the suggested six features in [TMM+18] who only applied the GRU Model. Furthermore, the SDN controller can retrieve all the considered features from the flow statistics features of the OpenFlow Switches.

For the evaluation of the methodologies above, the experiments follow the same criteria that subsection 5.3.2 introduces. Thereby, the experiments consider four measures (TP, TN, FP, and FN) to calculates the metrics: Accuracy, Precision, Recall, and F1 score. In the experiments, TensorFlow and Keras framework are used [Ker, AAB+16] to implement RNN, LSTM, and GRU in Python programming language. For selecting the best values of hyper-parameters and build the neural network structure. The experiment considers the best practice, trial, and error, or human knowledge [ELKJ20] alongside testing different values and recording the corresponding results in each test experiment. Consequently, the experiments identify the values providing the highest accuracy in trade-off with training time. All DLA models include two hidden layers with 32 and 16 neural nodes that apply ReLu activation function. The dense layer has two units as an output layer that work with the Sigmoid function. Additionally, the experiments use a Nadam optimizer [Doz16] and a MSE for the model. The hyper-parameter configuration is 100, 10, and 0.001 for the batch size, epoch, and learning rate. Table 5.3 shows the models' details for 48 and 6 features. The experiments were performed on Ubuntu 18.04.5 LTS 64-bit operating system with Intel®Core™ i5-8400, CPU @ 2.80GHz $\times$ 6, 16 GB of RAM, and AMD® Rv635 Graphics. In the end, the attack samples used during the testing phase has different distribution in the training phase. The data records were mixed and then divided 80% and 20% for training and testing, respectively. Firstly, the

	48 Features			6 Features		
Algorithm	*Input Layers*	*Hidden Layers*	*Output Layers*	*Input Layers*	*Hidden Layers*	*Output Layers*
RNN, LSTM, GRU	48	32, 16	2	6	6, 4, 2	2

Table 5.3. – Neural Network Model Structure

detection performance of RNN, LSTM, and GRU with 48 features are presented in terms of accuracy, precision, recall, f1 score. As shown in Table 5.4, all models achieve good results with around 10% enhancements in comparison to the models with 6 features. Moreover, LSTM shows the best results in term of precision and accuracy, although the training time is always faster for RNN and GRU according to Table 5.5. Secondly, the Receiver Operating Characteristic (ROC) curve is introduced for both numbers of features in order to estimate how correctly the models work. The ROC curve depicts the relation between false positive rate and true positive rate which provides the Area Underneath the ROC Curve (AUC) that is useful to define which classifier predicts the classes best. Therefore, Figure 5.25 show that our models with 48 features grant 13% AUC enhancement for LSTM, and around of 15% for both of RNN and GRU compared to the models with 6 features. Considering 48 features, LSTM is able to classify 98% of positive and negative classes successfully, while RNN and GRU are able to correctly classify 96.2% and 96.4% respectively.

	with 48 Features				with 6 Features			
Algorithm	*Precision*	*Recall*	*F1 Score*	*Accuracy*	*Precision*	*Recall*	*F1 Score*	*Accuracy*
RNN	0.97899	0.99658	0.98771	0.980946	0.89941	0.99577	0.94514	0.91117
LSTM	0.98842	0.99702	0.99270	0.98874	0.92130	0.98771	0.95335	0.92573
GRU	0.97948	0.99757	0.98844	0.98208	0.90175	0.99544	0.94628	0.91315

Table 5.4. – Detection Performance Comparison for All Attacks

In addition to the foreseen results, the experiments analyze different attacks and compare the models' performance with RNN, LSTM, and GRU. All models show outstanding performance in terms of accuracy. The precision is also perfect for the detection of DDoS, DoS, and Probe. Besides, LSTM provides the best result for detecting botnet, web-attack, brute force attacks, and U2R attacks compared to RNN and GRU, as seen in Table 5.5. Generally, all models consistently have good scores on all metrics for DDoS, DoS, and probe classes. But, the performance notably declined on the botnet, web attack, and U2R classes due to the low number of samples in the dataset. The models can detect these attacks better when they work with the entire dataset because

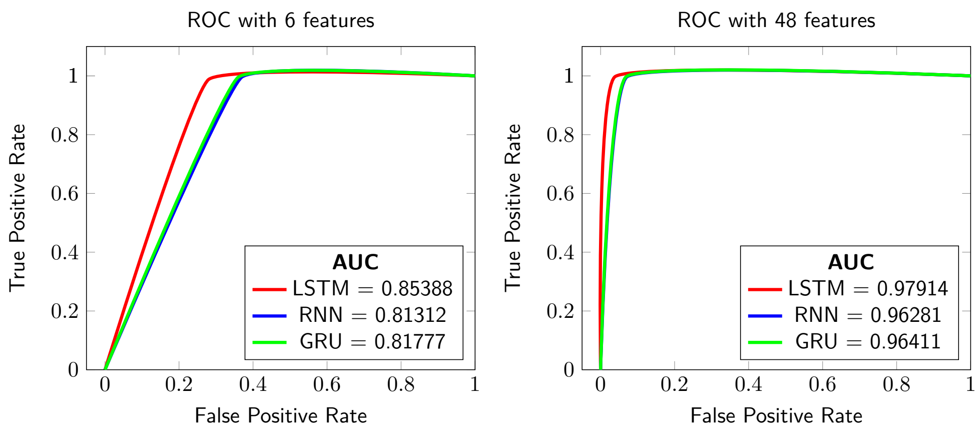


Figure 5.25. – ROC Curves.

Learning Model	*DDoS*	*DoS*	*Probe*	*Botnet*	*Web-Attack*	*BFA*	*U2R*	
Accuracy Score.								
LSTM	0.99940	0.98131	0.98192	0.99897	0.96633	0.99584	0.99802	
RNN	0.99936	0.98975	0.98333	0.99876	0.96320	0.99126	0.9989	
GRU	0.99943	0.98332	0.98189	0.99584	0.96342	0.99334	0.99948	
Precision Score.								
LSTM	0.99925	0.97793	0.97403	0.70212	0.07444	0.99476	0.074074	
RNN	0.99911	0.99536	0.97395	0.66000	0.068518	0.85972	0.13333	
GRU	0.99918	0.97565	0.97356	0.36666	0.06890	0.98958	0.28571	
Recall Score.								
LSTM	0.99959	0.97957	0.99628	1.00000	0.94871	0.67615	0.50000	
RNN	0.99966	0.98125	0.99842	1.00000	0.94871	0.67615	0.50000	
GRU	0.99972	0.98666	0.99587	1.00000	0.94871	0.67615	0.50000	
F1 score.								
LSTM	0.99942	0.97875	0.98189	0.8250	0.13805	0.80508	0.12903	
RNN	0.99938	0.98826	0.98603	0.79518	0.12780	0.75697	0.21052	
GRU	0.99945	0.98113	0.98483	0.53658	0.12847	0.80338	0.36363	
Training Times.								All Types
LSTM	21.2022	18.53204	24.26479	19.42383	19.30129	11.52263	19.59553	41.39506
RNN	14.70538	12.73099	16.70782	13.21762	13.37479	8.08346	13.3889	28.74049
GRU	21.13238	18.36072	23.72825	18.99140	19.18356	11.55337	11.51159	40.94993

Table 5.5. – Evaluation of LSTM, RNN, and GRU with 48 Features

the records of the attack classes often have mutual features. Besides, recall and f1 score for RNN and GRU are relatively low on web and U2R attacks. Finally, we compared the performance of LSTM, RNN, and RNN in terms of training time. Table 5.5 shows that GRU and RNN needed less training time than LSTM, whereas RNN is always the fastest model dueusing one gate, which makes it vulnerable to the problem of vanishing gradients. Moreover, GRU is faster than LSTM, but its results are comparable to the LSTM results or even better when the number of samples gets lower. However, with an increasing sample number, LSTM is performing better, although it needs more time than the others for training or responding because LSTM has three gates.

5.6 Discussion

The presented measurements demonstrate that the suggested IDS framework is realistic and can be applied. Moreover, the suggested IDS shows superiority over other benchmarking work regarding the attack's detection or countermeasures. Although the solution depends on three components working individually, the measurements introduce the enhancements in different aspects. Besides, the researchers can easily improve the performance of suggested IDS and add more innovations according to the administration requirements. For example, the used DLAs need more training on new attack types with different datasets.

Furthermore, the Major issue that limits the performance of the MLA/DLA is the overfitting problem. The model perfectly functions during the training time, but it is unable to show a great outcome with different data types due to several possible reasons. For example, the model complexity and the low amount of data samples used in training. Still, MLAs are data-hungry, which means they often request massive training data, which is another significant issue, particularly in the network security field. Also, the dataset's availability is subjected to several challenges, such as privacy or illegal issues, because this dataset has sensitive information to reveal to the public. The existing works in the SDN security area used the same distribution of testing data similar to the training. Therefore the evaluation of such methods is unreliable for anomaly detection since the simple algorithm would provide high accuracy. Besides, once the same model runs through zero-day attacks, it causes very high false rates and lousy performance. Based on this discussion, the best approach for examining the proposed IDS is to evaluate the IDS's ability to analyze data that were not used in the training time as this work investigate and successfully achieve [EJNJ21].

Chapter 6

Conclusions and Outlook

This chapter firstly concludes the dissertation and provides the main results. Afterward, an outlook manifests open issues and possible future extensions of this work.

6.1 Conclusions

This dissertation highlights the SDN concept, which has emerged as a new intelligent architecture to reduce the hardware limitations of traditional networks. The essential improvement of introducing SDN is dissociating the control plane outside the forwarding nodes and enabling external data management by a logical software component called the controller. Consequently, the SDN smoothly abstracts the components description, the network nodes functions, and the protocols responsible for controlling the forwarding devices. Thereby, the controller monitors network flow traffics, publishes administration policies, and handles errors based on the monitoring outcomes.

Furthermore, the dissertation concentrates on the security aspects. It proposes an IDS framework treating the vulnerabilities that the malicious user (the adversary) exploiting to launch DoS/DDoS attacks towards the SDN resource. The first vulnerability appears with the adversary's ability to exploit the Southbound API between the SDN controller and the OpenFlow switch. The adversary submits a massive number of spoofing packets towards the edge switch. But, the switch table has no flow rule matching them. Therefore, the switch forwards the crafting packets to the controller. Thus, the packets exhaust the switch and controller resources. In the second vulnerability, the adversary exploits the pre-installed instructions in the flow table. After the adversary obtains information about the existing flow rules, a submitted packet is forged to fit the flow rule. Hence, the switch forwards the malicious traffic toward the attack target.

An overview of the SDN environments has been introduced in chapter 2 with focuse on the SDN architecture, the tasks of the SDN layers, and the available APIs to ensure communication among them. Furthermore, the overview clarifies the superiority of the SDN concepts over the conventional network through demonstrating the SDN advantages. Thereafter, necessary technical details about the OpenFlow protocol are also provided. These details show the OpenFlow role, different OpenFlow versions, and the exchanging messages between the SDN controller and switches. Accordingly, chapter 2 explains the design of both the controller and switch.

Following an overview discussed the security threats and comprehensive view of the problem statement in chapter 3. The discussion illustrated the DDoS attack problem on its hazard on both SDN and legacy networks. Additionally, the SDN vulnerabilities are highlighted and the challenge value to prevent the adversary from exploiting them. Also, what makes the SDN environment is a perfect opportunity to address DDoS attacks. chapter 3 highlighted the state of the art to counter DDoS in SDN and classified the existing mechanisms according to four categories: Architecture-based solutions, Statistical-based solutions, Machine learning-based solutions, and Deep Learning-based solutions.

chapter 4 thoroughly presented the proposed IDS framework, its individual components and the relationship between the components. In addition, chapter 4 details clarified the relationship among the IDS components with the illumination of each component goal. Following this, chapter 4 also described each work that has been completed to achieve the proposed IDS framework individually. Firstly, it introduced the *Inspector* device and its goal and analyzed adding a new device to the SDN architecture. Also, the penetration probability of the *Inspector* is discussed. Then, the method to maintain the situation or isolate the Inspector. Secondly, the 1D-CNN is proposed based on entropy information as IDS to tackle the security issues of the SDN controller. Then, chapter 4 presented a new suggestion relying on the IPSec protocol concept. The countermeasure uses the AH method to prevent DoS and DDoS attacks and differentiate between a benign user and an attacker during the freeloading attack.

Moreover, a new dynamic window size methodology is suggested. However, detecting the DDoS attack (before applying the countermeasure in the third component) suffers from some drawbacks. It is a complex statistical process that depends on multiple tables and includes three attack types: TCP SYN, ICMP, and UDP flooding attacks. Therefore, the utilization of the third component from recurrent neural networks is also given.

The simulation and measurement parts investigated the performance of the suggested work in chapter 5, which provided preamble information about the simulation environments (Mininet), Ryu controller, and tools to launch the malicious traffic. Subsequently, chapter 5 gradually demonstrated the IDS components' evaluation. The evaluation considered the *Inspector* device in two phases; The *Inspector* device was first compared to one of the state-of-the-art work simulation studies modeled by Mininet emulator. The simulation concentrated on studying the impact of the *Inspector* on the network performance. The evaluation clarified the high ability of the *Inspector* to enhance the switch performance under the attack conditions in terms of switches' TCAM, OpenFlow Channel workload, and controller CPU and memory usages. In the second phase, the evaluation investigates the enhancements handling if the attacker controls the *Inspector* device to enable a malicious host to launch the DDoS attack towards the server. The evaluation showed that the enhancements do not negatively impact the OpenFlow channel workload or the controller's CPU usage. It also improved the target server in terms of server workload and CPU usage.

Moreover, the evaluation detailed the experimental setup and the used dataset (CI-CDDoS2019) to analyze the performance regarding the 1D-CNN model. The work compared to four MLA (ID3, random forest, Naïve Bayes, and logistic regression) proposed in state of the art. The result proved the superiority of the proposed model according to ROC, AUC, accuracy, precision, recall, and F1 score numbers. Afterward, the mininet setup topology to verify the model effectiveness as a fundamental unit in the second component of the proposed IDS. Also, the results showed the component's ability to detect the DDoS attack and produced good results compared to the attack condition in respect of TCAM, the controller's CPU usage, and OpenFlow channel bandwidth. Concerning the third component, chapter 5 detailed the implementation results and the simulations scenarios. Then, the section analyzed the effectiveness of the implemented attacks and the performance of the proposed countermeasures using IPSec protocol concepts. The result proved the enhancements under DDoS attack terms of packet delivery ratio, bandwidth rate, and delay time. Also, the result demonstrated the comparison to three previous works in detection time, 1-specificity, and bandwidth usage with the freeloading attack. In addition, chapter 5 explained the InSDN Dataset and evaluated the ability of the recurrent neural networks to detect both the controller injection inherited attacks in SDN. Afterward, the results compared to another existing approach, which revealed the advancements in ROC, AUC, accuracy, precision, recall, and F1 score numbers.

6.2 Outlook

Along with the progress of the dissertation work to investigate the particular problems and their corresponding solutions, several issues have been addressed. Therefore, additional investigations are recommended to improve the proposed IDS framework and make it better suited to the realization.

Regarding the proposed IDS framework, more investigations on the reality might be helpful to obtain more information about the method that the IDS should follow. Hence, the proposed IDS would have more accurate outcomes. Furthermore, the AI algorithms should be trained with other datasets and the model architectures could be enhanced by adding more functions or combining with other AI algorithms.

A part of the assumption is to deal with the Northbound API and the application layer. The complete IDS must consider the expected effects coming from the application layer and the relevant API. Although both have no standard and several projects started working to offer their fundamentals, the expected security threats are primarily obvious and applicable. Moreover, the IDS must consider the ability of the application layer to have complete knowledge about the whole network activities and domains. This way, part or complete IDS might be located on the application layer instead of the controller, which could help to reduce the controller workload.

The future work could depend on multiple controllers to manage the network. These controllers could work cloudy in a grid network. Consequently, the IDS framework uses the grid structure to avoid exploiting the penetrated controller. Therefore, the IDS framework must present SDN controllers that monitor the other controllers' performance or network domains to isolate the malicious controller.

Towards reaching the best network immunity against the security risks, the OpenFlow switch should be more intelligent to read the Cryptography's signs from the SDN controller or the equipped users. Moreover, the IPSec suites protocol could be modified to be more suitable for the SDN networks. Thus, the OpenFlow switch can differentiate between the benign and malicious packets without blocking the whole traffic coming from the malicious ports.

Appendices

Appendix A

Brief Description of the Mentioned Algorithms

The Algorithm	Brief Description
kNN	Stores the dataset without training. So, when new data comes, KNN compares it to the stored data to classify it according to a much similar category.
SVM	Finds a hyperplane in N-dimensional space (N, the number of features) to classifies the data based on the distance to the hyperplane.
BayesNet	Gives every attribute a probabilistic value (Weight). The result is the summation of the feature value multiply the probabilistic weight. The high result is the classification(e.g.,79% Winter, 21% Summer).
J48	Selects a certain attribute to be at the root node, and then J48 goes to the next branch for each possible attribute value. Finally, the leaf node has the classification result.
Random Tree	Feeds each specific tree by a different feature. Each tree will vote. the prediction with higher votes is the classification.
Logistic Regression	Uses sigmoid function $1/(1+e^{-x})$, if the output given by Sigmoid function is more than 0.5, the output is classified as 1. Otherwise, the output is classified as 0.
REPTree	Reduces the size of decision trees by removing parts of the tree that do not provide power to classify instances.
Fuzzy logic	Obtains a likelihood (Degrees of truth) for each input and produces outputs that depend on the states of the inputs.

RFA	Selects random subsets of variables for each tree and uses the most frequent tree output as the overall classification.
SOM	Each input is distributed to all nodes to be trained. The closest weight node to the current object becomes the winning or active unit, which provides the classification.
CNN	Arranges the available inputs to groups to apply some equations on every group individually. Thereby, arranges the result as new inputs to groups to apply some equations on every group individually. Repeat the same until obtaining the final result.
Exact-STORM	It calculates and stores the summary of the current window (a set of values). The stored summary contains the information of velocity of the *Packet_In* messages. The relevant information in the node such as the velocity of *Packet_In* messages, the identifier, and the number of succeeding neighbors of the node, and list that contains the identifiers of the most recent preceding neighbors.
BPNN	ANN algorithm has three layers: input layer, hidden layer, and output layer. In the output layer, If the outcome matches the expected outcome, it will be output. But, the error back-propagation starts, where the algorithm would adjust each layer's weight according to the gradient descent algorithm.
RNN, LSTM	ANN algorithms that use sequential or time-series data. They are DLAs used for ordinal or temporal problems. They learn based on training data. Moreover, their memory distinguishes them as they take information from prior inputs to influence the current input and output. Their output depends on the pre elements within the sequence. While future events would also help determine the output of a given sequence, unidirectional recurrent neural networks cannot account for these events in their prediction (RNN, LSTM, and GRU explained in section 4.5)

Table A.1. – Brief Description of the Mentioned MLAs and DLAs

Appendix B

The Selected Features for Recurrent Neural Networks

Feature Name	Description
Total Fwd Packets	The total number of packets sent
Protocol	The used connection protocol
Fwd IAT Min	The minimum time between two packets
Fwd IAT Max	The maximum time between two packets
Fwd IAT Mean	The mean time between two packets
Fwd IAT Std	The standard deviation time between two packets
Fwd IAT Total	The total time between two packets
Fwd Header Length	The total bytes used for headers
Total Length of Fwd Packet	The total size of the packet
Fwd Packet Length Min	The minimum size of packet
Fwd Packet Length Max	The maximum size of packet
Fwd Packet Length Mean	The Mean size of packet
Fwd Packet Length Std	The standard deviation size of packet
FWD Packets/s	Number of forward packets per second
Total Bwd Packets	The total number of packets
Bwd IAT Min	The minimum time between two packets
Bwd IAT Max	The maximum time between two packets
Bwd IAT Mean	The mean time between two packets
Bwd IAT Std	The standard deviation time between two packets

Bwd IAT Total	The total time between two packets
Bwd Header Length	The total bytes used for headers
Total Length of Bwd Packet	The total size of the packet
Bwd Packets/s	number of backward packets per second
Bwd Packet Length Min	The minimum size of packet
Bwd Packet Length Max	The maximum size of packet
Bwd Packet Length Mean	The mean size of packet
Bwd Packet Length Std	The standard deviation size of packet
Flow duration	The flow's duration measured in microseconds
Flow Bytes/s	number of flow bytes per second
Flow Packets/s	number of flow packets per second
Flow IAT Mean	The Mean time between two packets
Flow IAT Std	The standard deviation time between two packets
Flow IAT Max	The maximum time between two packets
Flow IAT Min	The minimum time between two packets
Packet Length Min	Minimum length of a packet
Packet Length Max	Maximum length of a packet
Packet Length Mean	Mean length of a packet
Packet Length Std	Standard deviation length of a packet
Packet Length Variance	Variance length of a packet
Average Packet Size	The packet average size
Active Min	Minimum time a flow was active before becoming idle
Active Mean	Mean time a flow was active before becoming idle
Active Max	Maximum time a flow was active before becoming idle
Active Std	Standard deviation time a flow was active before becoming idle
Idle Min	Minimum time a flow was idle before becoming active
Idle Mean	Mean time a flow was idle before becoming active
Idle Max	Maximum time a flow was idle before becoming active
Idle Std	Standard deviation time a flow was idle before becoming active

Table B.1. – The selected features for SDN environment

Appendix C

CICFlowMeter Features

Feature Name	Description
	In the forward direction
Total Fwd Packets	The total number of packets sent
Fwd IAT Min	The minimum time between two packets
Fwd IAT Max	The maximum time between two packets
Fwd IAT Mean	The mean time between two packets
Fwd IAT Std	The standard deviation time between two packets
Fwd IAT Total	The total time between two packets
Fwd PSH flags	Number of times the PSH flag was set in packets
Fwd URG Flags	Number of times the URG flag was set in packets
Fwd Header Length	The total bytes used for headers
Total Length of Fwd Packet	The total size of the packet
Fwd Packet Length Min	The minimum size of packet
Fwd Packet Length Max	The maximum size of packet
Fwd Packet Length Mean	The Mean size of packet
Fwd Packet Length Std	The standard deviation size of packet
FWD Packets/s	Number of forward packets per second
Fwd Segment Size Avg	Average size observed in the forward direction
Fwd Bytes/Bulk Avg	Average number of bytes bulk rate
Fwd Packet/Bulk Avg	Average number of packets bulk rate
Fwd Bulk Rate Avg	Average number of bulk rate
Subflow Fwd Packets	The average number of packets in a sub flow
Subflow Fwd Bytes	The average number of bytes in a sub flow

Fwd Init Win bytes	The total number of bytes sent in initial window
Fwd Act Data Pkts	Count of packets with at least 1 byte of TCP data payload
Fwd Seg Size Min	Minimum segment size observed
	In the backward direction
Total Bwd Packets	The total number of packets
Bwd IAT Min	The minimum time between two packets
Bwd IAT Max	The maximum time between two packets
Bwd IAT Mean	The mean time between two packets
Bwd IAT Std	The standard deviation time between two packets
Bwd IAT Total	The total time between two packets
Bwd PSH Flags	Number of times the PSH flag was set in packets
Bwd URG Flags	Number of times the URG flag was set in packets
Bwd Header Length	The total bytes used for headers
Total Length of Bwd Packet	The total size of the packet
Bwd Packets/s	Number of backward packets per second
Bwd Packet Length Min	The minimum size of packet
Bwd Packet Length Max	The maximum size of packet
Bwd Packet Length Mean	The mean size of packet
Bwd Packet Length Std	The standard deviation size of packet
Bwd Segment Size Avg	Average number of bytes bulk rate in the backward direction
Bwd Bytes/Bulk Avg	Average number of bytes bulk rate
Bwd Packet/Bulk Avg	Average number of packets bulk rate
Bwd Bulk Rate Avg	Average number of bulk rate
Subflow Bwd Packets	The average number of packets in a sub flow
Subflow Bwd Bytes	The average number of bytes in a sub flow
Bwd Init Win bytes	The total number of bytes sent in initial window
	Direction independent
Flow duration	The flow's duration measured in microseconds
Flow Bytes/s	Number of flow bytes per second
Flow Packets/s	Number of flow packets per second
Flow IAT Mean	The Mean time between two packets
Flow IAT Std	The standard deviation time between two packets
Flow IAT Max	The maximum time between two packets
Flow IAT Min	The minimum time between two packets
Packet Length Min	Maximum length of a packet
Packet Length Mean	Mean length of a packet

Packet Length Std	Standard deviation length of a packet
Packet Length Variance	Variance length of a packet
FIN Flag Count	Number of packets with FIN Flag
SYN Flag Count	Number of packets with SYN Flag
RST Flag Count	Number of packets with RST Flag
PSH Flag Count	Number of packets with PUSH Flag
ACK Flag Count	Number of packets with ACK Flag
URG Flag Count	Number of packets with URG Flag
CWR Flag Count	Number of packets with CWR Flag
ECE Flag Count	Number of packets with ECE Flag
Down/Up Ratio	Download and upload ratio
Average Packet Size	The packet average size
Active Min	Minimum time a flow was active before becoming idle
Active Mean	Mean time a flow was active before becoming idle
Active Max	Maximum time a flow was active before becoming idle
Active Std	Standard deviation time a flow was active before becoming idle
Idle Min	Minimum time a flow was idle before becoming active
Idle Mean	Mean time a flow was idle before becoming active
Idle Max	Maximum time a flow was idle before becoming active
Idle Std	Standard deviation time a flow was idle before becoming active

Table C.1. – CICDDoS2019 Features and their Description

Bibliography

[AAB$^+$16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[AFS] Abdullah Soliman Alshra'a, Ahmad Farahat, and Jochen Seitz. Deep Learning Algorithms for Detecting Denial of Service Attacks in Software Defined Networks. *Accepted: The 16th International Conference on Future Networks and Communications (FNC). August 9-12, 2021, Leuven, Belgium.*

[AGS18] Saleh Asadollahi, Bhargavi Goswami, and Mohammed Sameer. Ryu controller's scalability experiment on software defined networks. In *2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC)*, pages 1–5. IEEE, 2018.

[AHMYR$^+$17] Joaquin Alvarez-Horcajo, Isaias Martinez-Yelmo, Elisa Rojas, Juan A Carral, and Diego Lopez-Pajares. New cooperative mechanisms for software defined networks based on hybrid switches. *Transactions on Emerging Telecommunications Technologies*, 28(8):e3150, 2017.

[AMK$^+$18] Normaziah A Aziz, Teddy Mantoro, M Aiman Khairudin, et al. Software defined networking (SDN) and its security issues. In *2018 International Conference on Computing, Engineering, and Design (ICCED)*, pages 40–45. IEEE, 2018.

[AR18] Hatim Gasmelseed Ahmed and R Ramalakshmi. Performance Analysis of Centralized and Distributed SDN Controllers for Load Balancing

Application. In *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 758–764. IEEE, 2018.

[AS] Abdullah Soliman Alshra'a and Jochen Seitz. Towards Applying IPSec Between Edge Switches and End Users to Counter DDoS Attacks in SDNs. *The 19th IEEE International Conference on Smart City (SmartCity-2021), Haikou Hainan, China, December 20-22,2021.*

[AS19a] Abdullah Soliman Alshra'a and Jochen Seitz. External Device to Protect the Software-Defined Network Performance in Case of a Malicious Attack. In *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems*, pages 1–6, 2019.

[AS19b] Abdullah Soliman Alshra'a and Jochen Seitz. Using inspector device to stop packet injection attack in SDN. *IEEE Communications Letters*, 23(7):1174–1177, 2019.

[AS21] Abdullah Soliman Alshra'a and Jochen Seitz. One Dimensional Convolution Neural Network for Detection and Mitigation of DDoS Attacks in SDNs. *The 4th International Conference on Machine Learning for Networking (MLN'2021) Virtual conference, December 1-3, 2021*, pages –, 2021.

[ASAR16] Abdullah Aydeger, Nico Saputro, Kemal Akkaya, and Mohammed Rahman. Mitigating crossfire attacks using SDN-based moving target defense. In *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, pages 627–630. IEEE, 2016.

[ASS18] Abdullah Soliman Alshra'a, Parag Sewalkar, and Jochen Seitz. Enhanced failure recovery mechanism using openstate pipeline in SDN. In *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 104–109. IEEE, 2018.

[AT21] Neha Agrawal and Shashikala Tapaswi. An SDN-Assisted Defense Mechduanism for the Shrew DDoS Attack in a Cloud Computing Environment. *Journal of Network and Systems Management*, 29(2):1–28, 2021.

[BBMB16] Othmane Blial, Mouad Ben Mamoun, and Redouane Benaini. An overview on SDN architectures with multiple controllers. *Journal of Computer Networks and Communications*, 2016, 2016.

[BEB21] BEBA. BEhavioural BAsed Forwarding (BEBA) EU Project. `http://www.beba-project.eu/`, 2021. Online; accessed 19-JAN-2021.

[BEFEE16] Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. Software-defined networking (SDN): a survey. *Security and communication networks*, 9(18):5803–5833, 2016.

[BGB18] Jitendra Bhatia, Radha Govani, and Madhuri Bhavsar. Software defined networking: From theory to practice. In *2018 Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, pages 789–794. IEEE, 2018.

[Bha19] Dhruba Kumar Bhattacharyya. *DDoS attacks: evolution, detection, prevention, reaction, and tolerance*. Chapman and Hall/CRC, 2019.

[BM14] Wolfgang Braun and Michael Menth. Software-defined networking using OpenFlow: Protocols, applications and architectural design choices. *Future Internet*, 6(2):302–336, 2014.

[BS16] Narmeen Zakaria Bawany and Jawwad A Shamsi. Application layer DDoS attack defense framework for smart city using SDN. In *The Third International Conference on Computer Science, Computer Engineering, and Social Media (CSCESM2016)*, page 1, 2016.

[BS19] Naveen Bindra and Manu Sood. Detecting DDoS attacks using machine learning techniques and contemporary intrusion detection dataset. *Automatic Control and Computer Sciences*, 53(5):419–428, 2019.

[BSS17] Narmeen Zakaria Bawany, Jawwad A Shamsi, and Khaled Salah. DDoS attack detection and mitigation using SDN: methods, practices, and solutions. *Arabian Journal for Science and Engineering*, 42(2):425–441, 2017.

[CHC18] Yeim-Kuan Chang, Yi-Tsung Huang, and Yu-To Chen. An efficient label-based packet forwarding scheme in software defined networks. In *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 191–196. IEEE, 2018.

[CHL15] Chang Ching-Hao and Ying-Dar Lin. Openflow version roadmap. *Study, Dept. of Computer Science, National Chiao Tung University, Taiwan*, 2015.

[CIC] CICFlowMeter V3 Python Implementation. `https://pypi.org/project/cicflowmeter/`. Accessed: 2021-08-01.

[Cis20] Cisco. Cisco annual internet report (2018–2023) white paper. *Online](accessed March 26, 2021) https://www. cisco. com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/whitepaper-c11-741490. html*, 2020.

[CIV20] Juan Camilo Correa Chica, Jenny Cuatindioy Imbachi, and Juan Felipe Botero Vega. Security in SDN: A comprehensive survey. *Journal of Network and Computer Applications*, 159:102595, 2020.

[CJS$^+$16] Kuan-yin Chen, Anudeep Reddy Junuthula, Ishant Kumar Siddhrau, Yang Xu, and H Jonathan Chao. SDNShield: Towards more comprehensive defense against DDoS attacks on SDN control plane. In *2016 IEEE Conference on Communications and Network Security (CNS)*, pages 28–36. IEEE, 2016.

[CLMR19] Mauro Conti, Chhagan Lal, Reza Mohammadi, and Umashankar Rawat. Lightweight solutions to counter DDoS attacks in software defined networking. *Wireless Networks*, 25(5):2751–2768, 2019.

[CWLZ19] Jie Cui, Mingjun Wang, Yonglong Luo, and Hong Zhong. DDoS detection and defense mechanism based on cognitive-inspired computing in SDN. *Future generation computer systems*, 97:275–283, 2019.

[CYL$^+$16] Yunhe Cui, Lianshan Yan, Saifei Li, Huanlai Xing, Wei Pan, Jian Zhu, and Xiaoyang Zheng. SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks. *Journal of Network and Computer Applications*, 68:65–79, 2016.

[DAJ19] Shi Dong, Khushnood Abbas, and Raj Jain. A survey on distributed denial of service (DDoS) attacks in SDN and cloud computing environments. *IEEE Access*, 7:80813–80828, 2019.

[DDZX16] Ping Dong, Xiaojiang Du, Hongke Zhang, and Tong Xu. A detection method for a novel DDoS attack against SDN controllers by vast new low-traffic flows. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2016.

[DGL+19] Shuhua Deng, Xing Gao, Zebin Lu, Zhengfa Li, and Xieping Gao. DoS vulnerabilities and mitigation strategies in software-defined networks. *Journal of Network and Computer Applications*, 125:209–219, 2019.

[DGLG17] Shuhua Deng, Xing Gao, Zebin Lu, and Xieping Gao. Packet injection attack and its defense in software-defined networks. *IEEE Transactions on Information Forensics and Security*, 13(3):695–705, 2017.

[dOSSP14] Rogério Leão Santos de Oliveira, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete. Using Mininet for emulation and prototyping Software-Defined Networks. In *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2014.

[Doz16] Timothy Dozat. Incorporating Nesterov Momentum into Adam. `https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ`, 2016. Accessed: 2021-08-01.

[dSASZ17] Antonio Cleber de S. Araujo, Leobino N. Sampaio, and Artur Ziviani. BEEP: Balancing Energy, Redundancy, and Performance in Fat-Tree Data Center Networks. *IEEE Internet Computing*, 21(4):44–53, 2017.

[DSB21] Afsaneh Banitalebi Dehkordi, MohammadReza Soltanaghaei, and Farsad Zamani Boroujeni. The DDoS attacks detection through machine learning and statistical methods in SDN. *The Journal of Supercomputing*, 77(3):2383–2415, 2021.

[DSR17] Ahmed Dawoud, Seyed Shahristani, and Chun Raun. Software-defined network security: Breaks and obstacles. In *Networks of the Future: Architectures, Technologies, and Implementations*, pages 89–100. CRC Press, 2017.

[Eas21] William Easttom. Virtual Private Networks, Authentication, and Wireless Security. In *Modern Cryptography*, pages 299–317. Springer, 2021.

[EDP21] Lubna Fayez Eliyan and Roberto Di Pietro. DoS and DDoS attacks in Software Defined Networks: A survey of existing solutions and research challenges. *Future Generation Computer Systems*, 122:149–171, 2021.

[EJNJ21] Mahmoud Said Elsayed, Hamed Z Jahromi, Muhammad Mohsin Nazir, and Anca Delia Jurcut. The Role of CNN for Intrusion Detection Systems: An Improved CNN Learning Approach for SDNs. pages 91–104, 2021.

[ELKJ20] Mahmoud Said Elsayed, Nhien-An Le-Khac, and Anca D Jurcut. InSDN: A Novel SDN Intrusion Dataset. *IEEE Access*, 8:165263–165284, 2020.

[fC19] Canadian Institute for Cybersecurity. DDoS Evaluation Dataset (CIC-DDoS2019). `https://www.unb.ca/cic/datasets/ddos-2019.html`, 2019. Accessed: 2021-08-01.

[GAA+14] Kostas Giotis, Christos Argyropoulos, Georgios Androulidakis, Dimitrios Kalogeras, and Vasilis Maglaris. Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. *Computer Networks*, 62:122–136, 2014.

[GBC16] Paul Goransson, Chuck Black, and Timothy Culver. *Software defined networks: a comprehensive approach.* Morgan Kaufmann, 2016.

[GBCMVVLV20] Jesús Galeano-Brajones, Javier Carmona-Murillo, Juan F Valenzuela-Valdés, and Francisco Luna-Valero. Detection and mitigation of dos and ddos attacks in iot-based stateful sdn: An experimental approach. *Sensors*, 20(3):816, 2020.

[GD21] Mozhgan Ghasabi and Mahmood Deypir. Using optimized statistical distances to confront distributed denial of service attacks in software defined networks. *Intelligent Data Analysis*, 25(1):155–176, 2021.

[GTV+14] Riccardo Guerzoni, Riccardo Trivisonno, Ishan Vaishnavi, Zoran Despotovic, Artur Hecker, Sergio Beker, and David Soldani. A novel approach to virtual networks embedding for SDN management and orchestration. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–7. IEEE, 2014.

[HCA+20] A Hussein, Louma Chadad, Nareg Adalian, Ali Chehab, Imad H Elhajj, and Ayman Kayssi. Software-Defined Networking (SDN): the security review. *Journal of Cyber Security Technology*, 4(1):1–66, 2020.

[HJT+20] Tao Han, Syed Rooh Ullah Jan, Zhiyuan Tan, Muhammad Usman, Mian Ahmad Jan, Rahim Khan, and Yongzhao Xu. A comprehensive survey of security threats and their mitigation techniques for next-generation SDN controllers. *Concurrency and Computation: Practice and Experience*, 32(16):e5300, 2020.

[HM21] Yogita Hande and Akkalashmi Muddana. A survey on intrusion detection system for software defined networks (SDN). In *Research Anthology on Artificial Intelligence Applications in Security*, pages 467–489. IGI Global, 2021.

[HOC+20] Chuxiong Hu, Tiansheng Ou, Haonan Chang, Yu Zhu, and Limin Zhu. Deep GRU Neural Network Prediction and Feed Forward Compensation for Precision Multiaxis Motion Control Systems. *IEEE/ASME Transactions on Mechatronics*, 25(3):1377–1388, 2020.

[HSI+21] Mohammad Kamrul Hasan, Muhammad Shafiq, Shayla Islam, Bishwajeet Pandey, Yousef A Baker El-Ebiary, Nazmus Shaker Nafi, R Ciro Rodriguez, and Doris Esenarro Vargas. Lightweight Cryptographic Algorithms for Guessing Attack Protection in Complex Internet of Things Applications. *Complexity*, 2021, 2021.

[HTY+19] Muhammad Reazul Haque, Saw C Tan, Zulfadzli Yusoff, Ching K Lee, and Rizaludin Kaspin. DDoS attack monitoring using smart controller placement in software defined networking architecture. In *Computational Science and Technology*, pages 195–203. Springer, 2019.

[IIAR20] Md Tariqul Islam, Nazrul Islam, and Md Al Refat. Node to node performance evaluation through RYU SDN controller. *Wireless Personal Communications*, pages 1–16, 2020.

[JASD12] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first Workshop on Hot Topics in Software Defined Detworks*, pages 127–132, 2012.

[JFG+18] Feng Jiang, Yunsheng Fu, Brij B Gupta, Yongsheng Liang, Seungmin Rho, Fang Lou, Fanzhi Meng, and Zhihong Tian. Deep learning based multi-channel intelligent attack detection for data security. *IEEE transactions on Sustainable Computing*, 5(2):204–212, 2018.

[JZZC16] Yajie Jiang, Xiaoning Zhang, Quan Zhou, and Zijing Cheng. An entropy-based DDoS defense mechanism in software defined networks. In *International Conference on Communications and Networking in China*, pages 169–178. Springer, 2016.

[KAGA18] Kübra Kalkan, Levent Altay, Gürkan Gür, and Fatih Alagöz. JESS: Joint entropy-based DDoS defense scheme in SDN. *IEEE Journal on Selected Areas in Communications*, 36(10):2358–2372, 2018.

[KDA19] Prabhakar Krishnan, Subhasri Duttagupta, and Krishnashree Achuthan. VARMAN: Multi-plane security framework for software defined networks. *Computer Communications*, 148:215–239, 2019.

[Ker] Keras Framework. `https://keras.io/`. Accessed: 2021-08-01.

[Ket17] Nikhil Ketkar. Introduction to keras. In *Deep learning with Python*, pages 97–111. Springer, 2017.

[KK05] Stephen Kent and Seo Karen. Security Architecture for the Internet Protocol. Request for Comments 4301, Internet Engineering Task Force, December 2005.

[LBJ$^+$17] Madhusanka Liyanage, An Braeken, Anca Delia Jurcut, Mika Ylianttila, and Andrei Gurtov. Secure communication channel architecture for software defined mobile networks. *Computer Networks*, 114:32–50, 2017.

[LDGMG17] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of Tor Traffic Using Time Based Features. In *International Conference on Information Systems Security and Privacy (ICISSP 2017)*, pages 253–262, 2017.

[LMMLPG19] Gabriel Lopez-Millan, Rafael Marin-Lopez, and Fernando Pereniguez-Garcia. Towards a standard SDN-based IPsec management framework. *Computer Standards & Interfaces*, 66:103357, 2019.

[LSL$^+$20] Zohaib Latif, Kashif Sharif, Fan Li, Md Monjurul Karim, Sujit Biswas, and Yu Wang. A comprehensive survey of interface protocols for software defined networks. *Journal of Network and Computer Applications*, 156:102563, 2020.

[LWY+18] Chuanhuang Li, Yan Wu, Xiaoyong Yuan, Zhengjun Sun, Weiming Wang, Xiaolin Li, and Liang Gong. Detection and defense of DDoS attack–based on deep learning in OpenFlow-based SDN. *International Journal of Communication Systems*, 31(5):e3497, 2018.

[Lyo14] Gordon Lyon. Nmap security scanner. *línea] URL: http://nmap.org/*, 2014. Accessed: 2021-08-01.

[M+11] Wes McKinney et al. pandas: a foundational Python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011.

[MBES16] Mohammad Mousa, Ayman M Bahaa-Eldin, and Mohamed Sobh. Software Defined Networking concepts and challenges. In *2016 11th International Conference on Computer Engineering & Systems (ICCES)*, pages 79–90. IEEE, 2016.

[Min21] Mininet emulator. `http://www.mininet.org/`, 2021. Online; accessed 19-JAN-2021.

[MJC17] Reza Mohammadi, Reza Javidan, and Mauro Conti. SLICOTS: An SDN-based lightweight countermeasure for TCP SYN flooding attacks. *IEEE Transactions on Network and Service Management*, 14(2):487–497, 2017.

[MMS19] Pedro Manso, José Moura, and Carlos Serrão. SDN-based intrusion detection system for early detection and mitigation of DDoS attacks. *Information*, 10(3):106, 2019.

[MOKKV19] Myo Myint Oo, Sinchai Kamolphiwong, Thossaporn Kamolphiwong, and Sangsuree Vasupongayya. Advanced support vector machine-(ASVM-) based detection for distributed denial of service (DDoS) attack on software defined networking (SDN). *Journal of Computer Networks and Communications*, 2019, 2019.

[Mon18] Robert Montante. Using Scapy in Teaching Network Header Formats: Programming Network Headers for Non-Programmers. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 1106–1106. ACM, 2018.

[MR17] Deep Medhi and Karthik Ramasamy. *Network routing: algorithms, protocols, and architectures.* Morgan Kaufmann, 2017.

[NCLP20] Matheus P Novaes, Luiz F Carvalho, Jaime Lloret, and Mario Lemes Proença. Long short-term memory and fuzzy logic for anomaly detection and mitigation in software-defined network environment. *IEEE Access*, 8:83765–83781, 2020.

[Nip21] Nippon Telegraph and Telephone Corporation. Ryu Documentation: OpenFlow v1.3 Messages and Structures. `https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html`, 2021. [Online; accessed 1-August-2021].

[NPK$^+$18] Tran Manh Nam, Phan Hai Phong, Tran Dinh Khoa, Truong Thu Huong, Pham Ngoc Nam, Nguyen Huu Thanh, Luong Xuan Thang, Pham Anh Tuan, Vu Duy Loi, et al. Self-organizing map-based approaches in DDoS flooding detection using SDN. In *2018 International Conference on Information Networking (ICOIN)*, pages 249–254. IEEE, 2018.

[ONF] ONF Open Networking foundation. `https://opennetworking.org/sdn-definition/`. Online, Accessed: 2021-08-01.

[Opea] https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.0.pdf. `https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.0.pdf`. Accessed: 2021-04-28.

[Opeb] OpenFlow Switch Consortium and Others. OpenFlow Switch Specification Version 1.1.0. 2011. `https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.1.0.pdf`. Accessed: 2021-04-28.

[Opec] OpenFlow Switch Consortium and Others. OpenFlow Switch Specification Version 1.2.0. 2011. `https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.2.pdf`. Accessed: 2021-04-28.

[Oped] OpenFlow Switch Consortium and Others. OpenFlow Switch Specification Version 1.3.0. 2012. `https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf`. Accessed: 2021-04-28.

[Opee] OpenFlow Switch Consortium and Others. OpenFlow Switch Specification Version 1.4.0. 2013. `https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf`. Accessed: 2021-04-28.

[Opef] OpenFlow Switch Consortium and Others.OpenFlow Switch Specification Version 1.0.0. 2009 . `https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf`. Accessed: 2021-08-01.

[PBP16] Trung V Phan, Nguyen Khac Bao, and Minho Park. A novel hybrid flow-based handler with DDoS attacks in software-defined networking. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pages 350–357. IEEE, 2016.

[PCK16] Younghee Park, Sang-Yoon Chang, and Lavanya M Krishnamurthy. Watermarking for detecting freeloader misbehavior in software-defined networks. In *2016 International conference on computing, networking and communications (ICNC)*, pages 1–6. IEEE, 2016.

[PDFN17] Túlio A Pascoal, Yuri G Dantas, Iguatemi E Fonseca, and Vivek Nigam. Slow TCAM exhaustion DDoS attack. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 17–31. Springer, 2017.

[PP18] Aditya Prakash and Rojalina Priyadarshini. An intelligent software defined network controller for preventing distributed denial of service attack. In *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, pages 585–589. IEEE, 2018.

[PP19] Mahesh Kumar Prasath and Balasubramani Perumal. A meta-heuristic Bayesian network classification for intrusion detection. *International Journal of Network Management*, 29(3):e2047, 2019.

[PPS19] Manaswi Parashar, Amarjeet Poonia, and Kandukuru Satish. A Survey of Attacks and their Mitigations in Software Defined Networks.

In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–8. IEEE, 2019.

[Psu] Psutil. `https://pypi.org/project/psutil/`. Online; accessed 19-JAN-2021.

[RBN$^+$19] Filippo Rebecchi, Julien Boite, Pierre-Alexis Nardin, Mathieu Bouet, and Vania Conan. DDoS Protection with Stateful Software-Defined Networking. *International Journal of Network Management*, 29(1), 2019.

[RR16] Danda B Rawat and Swetha R Reddy. Software defined networking architecture, security and energy efficiency: A survey. *IEEE Communications Surveys & Tutorials*, 19(1):325–346, 2016.

[San06] S Sanfilippo. Hping Security Tool. `http://www.hping.org`, 2006. Online, Accessed: 2021-08-01.

[SB16] Radhika Sukapuram and Gautam Barua. PPCU: Proportional per-packet consistent updates for software defined networks. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pages 1–2. IEEE, 2016.

[SB20] Jagdeep Singh and Sunny Behal. Detection and mitigation of DDoS attacks in SDN: A comprehensive review, research challenges and future directions. *Computer Science Review*, 37:100279, 2020.

[Sca] Packet crafting for Python2 and Python3. `https://scapy.net/`. Online; Accessed: 2021-08-01.

[SDR19] Rochak Swami, Mayank Dave, and Virender Ranga. Defending DDoS against software defined networks using entropy. In *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pages 1–5. IEEE, 2019.

[SDR21] Rochak Swami, Mayank Dave, and Virender Ranga. DDoS attacks and defense mechanisms using machine learning techniques for SDN. In *Research Anthology on Combating Denial-of-Service Attacks*, pages 248–264. IGI Global, 2021.

[SF18] Nishant Shukla and Kenneth Fricklas. *Machine learning with TensorFlow*. Manning Shelter Island, Ny, 2018.

[SKBJ20] Arash Shaghaghi, Mohamed Ali Kaafar, Rajkumar Buyya, and Sanjay Jha. Software-defined network (SDN) data plane security: issues, solutions, and future directions. *Handbook of Computer Networks and Cyber Security*, pages 341–387, 2020.

[SLHG19] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A Ghorbani. Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–8. IEEE, 2019.

[SPT$^+$18] Kshira Sagar Sahoo, Deepak Puthal, Mayank Tiwary, Joel JPC Rodrigues, Bibhudatta Sahoo, and Ratnakar Dash. An early detection of low rate DDoS attack to SDN based data center networks using information distance metrics. *Future Generation Computer Systems*, 89:685–697, 2018.

[SSC$^+$13] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. OpenFlow: Meeting Carrier-Grade Recovery Requirements. *Computer Communications*, 36(6):656–665, 2013.

[SSS$^+$20] Reneilson Santos, Danilo Souza, Walter Santo, Admilson Ribeiro, and Edward Moreno. Machine learning algorithms to detect DDoS attacks in SDN. *Concurrency and Computation: Practice and Experience*, 32(16):e5402, 2020.

[Sta19] RD Statista. Internet of Things-Number of connected devices worldwide 2015-2025. *Statista Research Department. statista. com/statistics/471264/iot-numberof-connected-devices-worldwide*, 2019.

[SYPG13] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 413–424, 2013.

[TMM$^+$18] Tuan A Tang, Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi, and Mounir Ghogho. Deep recurrent neural network for intrusion detection in sdn-based networks. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 202–206. IEEE, 2018.

[Too07] Kali Tools. The CAIDA UCSD "DDoS Attack 2007" Dataset. `https://www.caida.org/catalog/datasets/ddos-20070804_datase`, 2007. Online, Accessed: 2021-08-01.

[Too19] Kali Tools. hping3 package description. `https://tools.kali.org/`, 2019. Online, Accessed: 2021-08-01.

[UJ18] Tushar Ubale and Ankit Kumar Jain. Taxonomy of DDoS attacks in software-defined networking environment. In *International Conference on Futuristic Trends in Network and Communication Technologies*, pages 278–291. Springer, 2018.

[UJ20] Tushar Ubale and Ankit Kumar Jain. Survey on DDoS attack techniques and solutions in software-defined network. In *Handbook of computer networks and cyber security*, pages 389–419. Springer, 2020.

[uRWA+20] Raihan ur Rasool, Hua Wang, Usman Ashraf, Khandakar Ahmed, Zahid Anwar, and Wajid Rafique. A survey of link flooding attacks in software defined network ecosystems. *Journal of Network and Computer Applications*, page 102803, 2020.

[WC17] Tao Wang and Hongchang Chen. SGuard: A lightweight SDN safe-guard architecture for DoS attacks. *China Communications*, 14(6):113–125, 2017.

[WGH+14] An Wang, Yang Guo, Fang Hao, TV Lakshman, and Songqing Chen. Scotch: Elastically scaling up sdn control-plane using vswitch based overlay. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 403–414, 2014.

[WHT+19] Yang Wang, Tao Hu, Guangming Tang, Jichao Xie, and Jie Lu. SGS: Safe-guard scheme for protecting control plane against DDoS attacks in software-defined networking. *IEEE Access*, 7:34699–34710, 2019.

[WJJ15] Rui Wang, Zhiping Jia, and Lei Ju. An entropy-based distributed DDoS detection mechanism in software-defined networking. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 310–317. IEEE, 2015.

[WL20] Lu Wang and Ying Liu. A DDoS Attack Detection Method Based on Information Entropy and Deep Learning in SDN. In *2020 IEEE*

4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), volume 1, pages 1084–1088. IEEE, 2020.

[WXG15] Haopei Wang, Lei Xu, and Guofei Gu. Floodguard: A dos attack prevention extension in software-defined networks. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 239–250. IEEE, 2015.

[WZLH15] Bing Wang, Yao Zheng, Wenjing Lou, and Y Thomas Hou. DDoS attack protection in the era of cloud computing and software-defined networking. *Computer Networks*, 81:308–319, 2015.

[XKL+18] Yang Xin, Lingshuang Kong, Zhi Liu, Yuling Chen, Yanmiao Li, Hongliang Zhu, Mingcheng Gao, Haixia Hou, and Chunhua Wang. Machine learning and deep learning methods for cybersecurity. *IEEE Access*, 6:35365–35381, 2018.

[XWX20] Jianfeng Xu, Liming Wang, and Zhen Xu. An enhanced saturation attack and its mitigation mechanism in software-defined networking. *Computer Networks*, 169:107092, 2020.

[YNDT18] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629, 2018.

[YWLW20] Meng Yue, Huaiyuan Wang, Liang Liu, and Zhijun Wu. Detecting DoS attacks based on multi-features in SDN. *IEEE Access*, 8:104688–104700, 2020.

[YXT+12] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi. SDNi: A Message Exchange Protocol for Software Defined Networks (SDNs) across Multiple Domains. `https://www.bibsonomy.org/bibtex/216048b64b0994fd6576585efe239a092/chesteve`, June 2012. Internet Draft.

[YY15] Qiao Yan and F Richard Yu. Distributed denial of service attacks in software-defined networking with cloud computing. *IEEE Communications Magazine*, 53(4):52–59, 2015.

[ZZ20] Yong Zhu and Dapeng Zhou. Security Technology of Wireless Sensor Network Based on IPSEC. In *The International Conference on Cyber Security Intelligence and Analytics*, pages 92–97. Springer, 2020.

List of Figures

List of Tables

Abbreviations

$1D - CNN$	One-Dimensional Convolutional Neural Networks
ACL	Access Control List
AH	Authentication Header
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
AUC	Underneath ROC Curve
BFA	Password Brute-Force Attack
$BPNN$	Back-Propagation Neural Networks
$CAIDA$	Center for Applied Internet Data Analysis
CLI	Command Line Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
$CRUD$	Create, Read, Update and Delete
$DDoS$	Distributed Denial of Service
$DHCP$	Dynamic Host Configuration Protocol

DLA	Deep Learning algorithm
DNS	Domain Name System
DoS	Denial of Service
DPID	Data Path Identifier
dst	Destination
EBD	Entropy Based Detection
EDDM	Entropy-based DDoS Defence Mechanism
ESP	Encapsulation Security Payload
EWMA	Exponentially Weighted Moving Average
FN	False Negatives
FP	False Positives
FSM	Finite State Machine
FTP	File Transfer Protocol
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HTTP	Hypertext Transfer Protocol
HTTPs	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
*ICMPV*6	Internet Control Message Protocol for IPv6
IDS	Intrusion Detection System
IKE	Internet Key Exchange
Inport	Input Port
IoT	Internet of Things

IP	Internet Protocol
IPSec	Internet Protocol Security
*IPV*6	Internet Protocol for version 6
JESS	Joint Entropy-based DDoS Defense Scheme
KNN	K-Nearest Neighbor
LDAP	Lightweight Directory Access Protocol
LSTM	Long Short-Term memory
MAC	Media Access Control
MLA	Machine Learning Algorithm
MPLS	Multiprotocol Label Switching
MSE	Mean Squared Error
MSSQL	Microsoft SQL Server
NFV	Network Functions Virtualization
NTP	Network Time Protocol
ONF	Open Networking Foundation
Outport	Output Ports
OXM	OpenFlow Extensible Match
QoS	Quality of Service
ReLU	Rectified Linear Unit
REST	Representational State Transfer
RFA	Random Forest Algorithm
RNN	Recurrent Neural Network

ROC	Receiver Operating characteristics
SDN	Software-Defined Networking
SDNi	Software-Defined Networking Interface
SNMP	Simple Network Management Protocol
SOM	Self Organization Map
SPRT	Sequential Probability Ratio Test
src	Source
SSDP	Simple Service Discovery Protocol
SSH	Secure Shell
SVM	Support Vector Machine
TCAM	Ternary Content-Addressable Memory
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
TLS	Transport Layer Security
TN	True Negative
ToS	Type of Service
TP	True Positive
U2R	Exploitation User-to-Root
UDP	User Datagram Protocol
UPnP	Universal Plug and Play
VLAN	Virtual LAN