DETECTION AND COUNTERMEASURE OF SATURATION ATTACKS IN SOFTWARE-DEFINED NETWORKS

by

Samer Yousef Khamaiseh

A dissertation

submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy in Computing

Boise State University

December 2019

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the dissertation submitted by

Samer Yousef Khamaiseh

Dissertation Title:	Detection and Countermeasure of Saturation Attacks in Software-Defined Networks
Date of Final Oral Examination:	25 October 2019

The following individuals read and discussed the dissertation submitted by student Samer Yousef Khamaiseh, and they evaluated the student's presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Dianxiang Xu, Ph.D.	Chair, Supervisory Committee
Edoardo Serra, Ph.D.	Co-Chair, Supervisory Committee
Jyh-haw Yeh, Ph.D.	Member, Supervisory Committee
Min Long, Ph.D.	Member, Supervisory Committee

The final reading approval of the dissertation was granted by Dianxiang Xu, Ph.D., Chair of the Supervisory Committee. The dissertation was approved by the Graduate College.

DEDICATION

Dedicated to My Parents and My Wife

ACKNOWLEDGMENTS

I did not find a single word that can depict my deepest and heartiest gratefulness, respectfulness, and thankfulness to my life teacher and research advisor Prof. Dianxiang Xu. During my Ph.D. journey, Prof. Xu acted as a life teacher by discussing my professional development and providing the space to improve my critical life-career skills which help me to have a clear path for me to follow. Prof. Xu patiently listened to my challenges and pushed me to address them.

As a Ph.D. advisor, Prof. Xu provided significant advice, assistance, confidence, and the space that helped me to accomplish my Ph.D. successfully. This research work could not be accomplished without the extraordinary guidance of Prof. Xu. Prof. Xu gave the needed inspiration to be a successful researcher and teacher in the future. There is nothing I can give to repay Prof. Xu suitably. Thus, I will cherish your lessons and hopefully, I will be an inspirational professor to others like you to me.

I'd like to express my deepest thankfulness and gratefulness to my Co-advisor Dr. Edaordo Serra for his deep collaboration and strong support. Dr. Serra dedicated a large amount of effort and time to improve my research work by providing extraordinary advice and guidance. Besides, Dr. Serra helped me in improving my research and teaching skills by providing the needed inspiration and guidelines.

I also, insistently and truly, wish to thank my parents, "Yousef and Amenah", and my sincere wife "Diana Cesar" for their unlimited support, encouragement, and assistance throughout my Ph.D. journey. They have told me that I can accomplish anything that I have set out to achieve. My parents have also taught me to be patient and committed to what I endeavor to accomplish. My parents and my wife are the main reason for any personal success that I have had and that I will have in the future. Many thanks to my sisters and brothers for their help and unlimited willingness to maintain their support and interest in my studies abroad.

A big thanks to Dr. Zhiyuan Li for his collaboration in this work. Special thanks go to my friends Dr. Izzat Al-Smadi and Zaid Al-Omari, for standing with me through this process, for providing unlimited help, support, and encouragement, and that they have always trusted in my capabilities in combination with pushing me to achieve my goals.

A big thanks goes to my committee members for their allocated time and suggestions, and many thanks go to the Department of Computer Science for its support, it has been my honor to be a Ph.D. student in this department at Boise State University.

ABSTRACT

The decoupling of control and data planes in software-defined networking (SDN) facilitates orchestrating the network traffic. However, SDN suffers from critical security issues, such as DoS saturation attacks on the data plane. These attacks can exhaust the SDN component resources, including the computational resources of the control plane, create a high packet loss rate and a long delay in delivering the OpenFlow messages due to the bandwidth consumption of the OpenFlow connection channel, and exhausting the buffer memory of the data plane.

Currently, most of the existing machine learning detection methods rely on a predefined time-window to start analyzing the network traffic to detect the saturation attacks caused by TCP-SYN flooding. However, saturation attacks range in duration, and a long-lasting attack can affect the entire SDN network. Therefore, if the time window is too large, the detection method response time will be long, and the attack may have an opportunity to saturate the network. If the time window is too small, the amount of the traffic may be insufficient to provide reliable detection results and the detection method will start frequently, which may cause a huge performance overhead for the SDN environment. Thus, identifying the proper time window for running the detection method and analyzing the traffic is a key concern.

For saturation attacks, the adoption of machine learning detection systems in the "real world" has been very limited. This is partly because of their deficiencies in detecting unknown saturation attacks. An unknown attack is an attack which is not represented in the dataset used to train the attack detection model. Therefore, evaluating the detection performance of the state-of-the-art supervised machine learning and semi-supervised algorithms on unknown saturation attacks is another key concern.

Furthermore, many of the proposed anomaly defense systems are deficient in mitigating the unknown saturation attacks and involve techniques which may not be compatible with OpenFlow protocol, such as modifying the data plane by adding extra devices, migrating the network traffic to a scrubbing center, and/or require extensive computational resources. Thus, an effective solution that is capable of detecting and mitigating known and unknown saturation attacks is an urgent need.

In this dissertation, we propose a defense framework to mitigate known and unknown saturation attacks for SDN. It resides on the application layer and can protect the computational resources of the control plane and data plane. The proposed defense system combines (1) a saturation attack detection module that is capable of detecting both known and unknown saturation attacks by leveraging the proper time window of OpenFlow traffic analysis combined with machine learning to identify the attacks, (2) a victim switch detection module that can detect and identify the victim of OpenFlow switches when they are targeted by known and unknown saturation attacks, and (3) a countermeasure module that can mitigate a family of saturation attacks and return the data plane settings to the pre-attack ones.

Implementation and experimental results demonstrate that, in comparison with the state-of-the-art defense systems, the proposed system provides effective protection for the SDN network — control plane, data plane, and OpenFlow connection channel — without extensive control plane computational resources and data plane flow table utilization.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

SDN	Software-Defined Networking
OF	OpenFlow Protocol
DoS	Denial of Service Attacks
DDoS	Distributed Denial of Service Attacks
IDS	Intrusion Detection System
SVM	Support Vector Machine
KNN	K-Nearest neighbors
NB	Naïve Bayes
ANN	Artificial Neural Network
SOM	Self-Organizing Map
DNN	Deep Neural Network
DT	Decision Tree
ML	Machine Learning
DL	Deep Learning

CHAPTER ONE: INTRODUCTION

1.1 SDN and OpenFlow

In traditional networks, the network administrator needs to configure the network devices to change the route of the traffic packets because control is distributed among all the network devices. The SDN offers a new way of managing and controlling networks by separating the control plane and the data plane. Figure 1 shows the basic architecture of the SDN environment, which is composed of a data plane and a control plane communicating through the southbound API. Above the control plane, the application layer resides, which comprises the business applications that communicate with the control plane via the northbound API.

The data plane includes the network hardware components: switches (e.g. OpenFlow switches) and routers which are responsible for forwarding operations. The OpenFlow switch consists of multiple flow-tables as a buffer to hold the flow-rule that controls the traffic.

The southbound API represents the interface between the network switches and the SDN controller. Basically, it allows the SDN controller to control the behavior of hardware devices in the SDN-network. The OpenFlow protocol is the standard and the most widely used southbound API.

The control plane includes the SDN controller, which is the brain of the network that orchestrates the entire network. The controller is a centralized controlling unit that translates the SDN applications' network requirements down to the data plane. It also provides the network information, such as network topology and statistical reports, to the business application that resides in the application layer. The communication between the business applications and the SDN controller travels via the northbound APIs.

The northbound APIs provide an abstraction of the network functions and enable the network applications and orchestration systems to dictate the behavior of the SDN network by providing a programmable interface to request the network services and dynamically configure the network.

OpenFlow is the first proposed communication protocol between the data plane and the control plane and has been defined as the standard southbound API used in the SDN architecture by the Open Network Foundation (ONC) [2]. According to the OpenFlow protocol, the OpenFlow switch consists of flow tables, group tables and an OpenFlow channel that provides the connection channel to exchange the OpenFlow messages between the SDN controller and OpenFlow switches.

The OpenFlow protocol has three types of messages. First, Control-to-switch messages are sent by the controller to update, add, or delete group/flow entries or request the status of switches. Second, Asynchronous messages are initiated by the OpenFlow switch and sent to the controller. These include Packet-In messages to inform the controller about a new packet arrival that does not match the flow entry rules or about changes in the switch state. Third, Symmetric messages are initiated in both directions from controller-to-switch or from switch-to-controller. These messages, such as a Hello-Message, are used to test the connection between the controller and switch and make sure that the connection is still alive.

The SDN hardware components are less expensive than the traditional network components since they do not need to be changed over time to upgrade the network. Because they are programmable – controlled by the SDN controller via the southbound API – they have a relatively long shelf-life. Conversely, the traditional network is made up of multiple connected switches that control the entire network, each of which needs to be managed and configured independently. As a result, any change, such as installing a new network application and/or changing the forwarding traffic rules of the network, needs human intervention and may take days or weeks to complete. This is because few APIs are exposed by a traditional network. Thus, the cost of upgrading and managing the hardware of a traditional network is higher than for an SDN, both in terms of dollars and time [1].

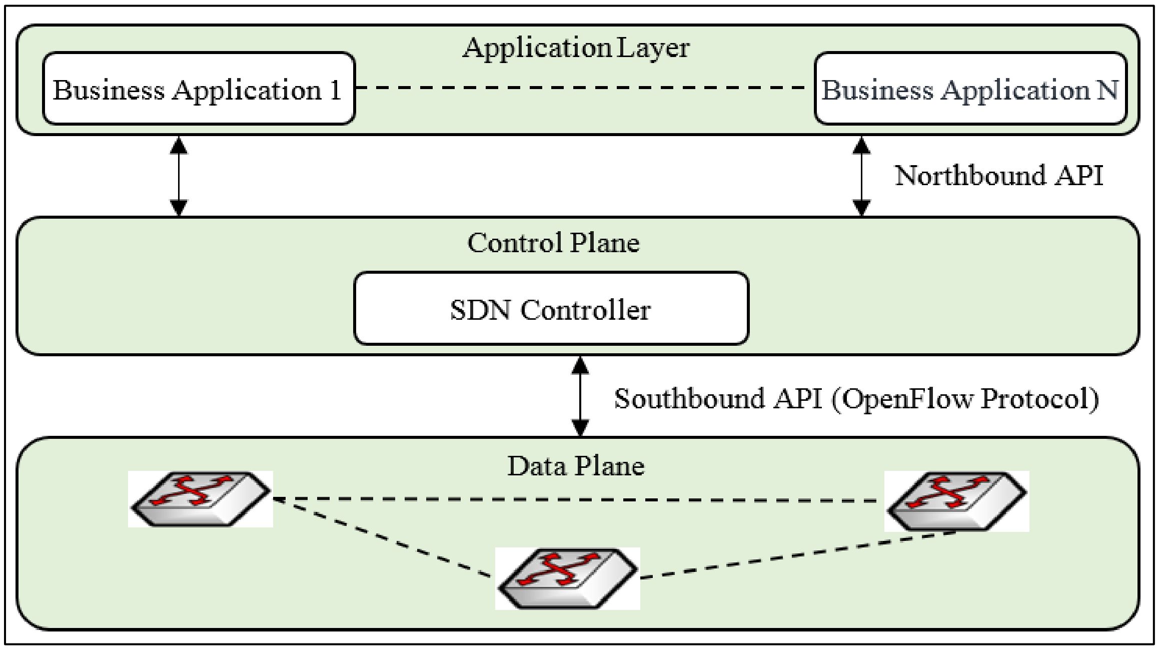


Figure 1 SDN Architecture

1.2 Saturation Attacks

When a new packet does not match any of the local flow-rules of the OpenFlow switch, a table-miss occurs. At this point, a Packet-In message will be generated, which contains the header of the table-miss packet if the switch buffer is not full. However, if the switch buffer is full, the whole table-miss packet will be encapsulated in the Packet-In message and sent to the controller. After receiving the Packet-In message, the controller will decide how to process the table-miss packet by sending Packet-Out and Packet-Mod messages to install flow-rule(s) in the switch flow table. This reactive packet processing approach of the OpenFlow network exposes a security vulnerability.

As depicted in Figure 2, a table-miss can be exploited by an attacker to consume the computation resources (e.g., CPU, memory) of the controller and switches and saturate the OpenFlow connection channel that is responsible for delivering the forwarding messages between the controller and the OpenFlow switches. Basically, an attacker can employ the TCP-SYN, UDP, ICMP, IP-Spoofing, TCP-SARFU flooding attacks, or their combinations (i.e., hybrid saturation attacks) to launch data-to-control plane attacks. This is accomplished by controlling many of the SDN network hosts (zombie machines) and sending a large number of forged packets to make it impossible to match any of the targeted OpenFlow switches' flow-rules. Thus, a large number of Packet-In messages are forwarded to the controller. Such a data-to-control plane attack exhausts the computation resources of the controller, as shown in Figure 3.

When a data-to-control plane flooding attack occurs, the controller will send a large number of Packet-Out and Packet-Mod messages, which will lead to a control-to-data plane flooding attack. Therefore, the targeted switch flow tables will be filled with fake flow-rules, which prevents the benign flow-rules from being installed. At this point, the victim switch buffer will be consumed, and it will not be able to process the legitimate new packets. Also, the OpenFlow channel bandwidth will be exhausted, which disables the delivery of OpenFlow messages between the controller and the OpenFlow switches, as shown in Figure 4.

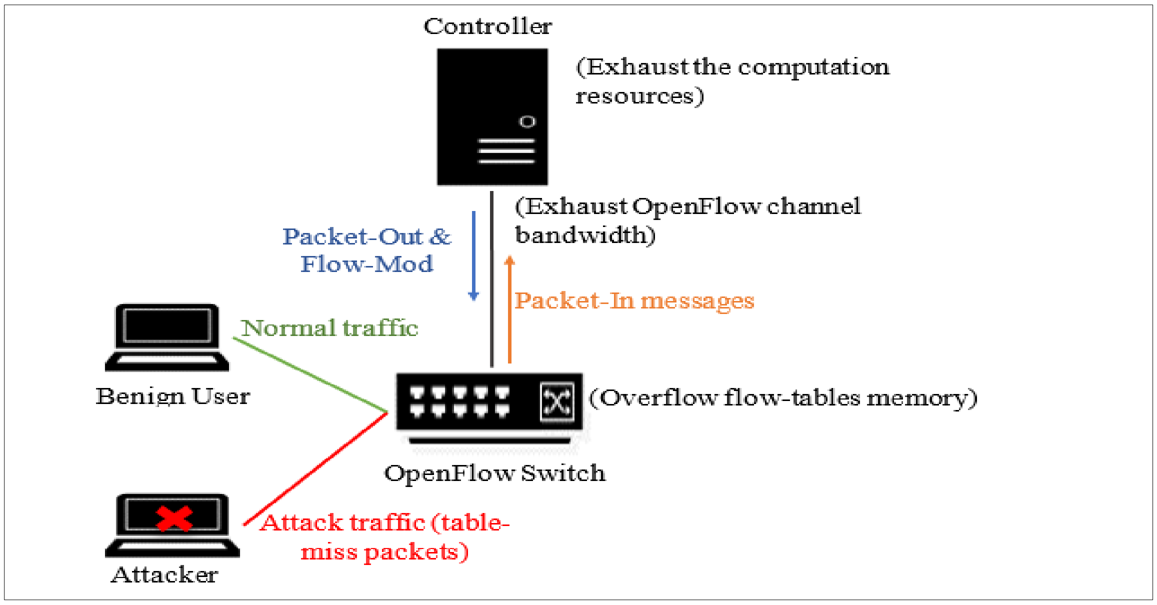

Controller
(Exhaust the computation resources)
(Exhaust OpenFlow channel bandwidth)
Packet-Out & Flow-Mod
Packet-In messages
Normal traffic
Benign User
(Overflow flow-tables memory)
OpenFlow Switch
Attack traffic (table-miss packets)
Attacker

Figure 2 Adversary Model

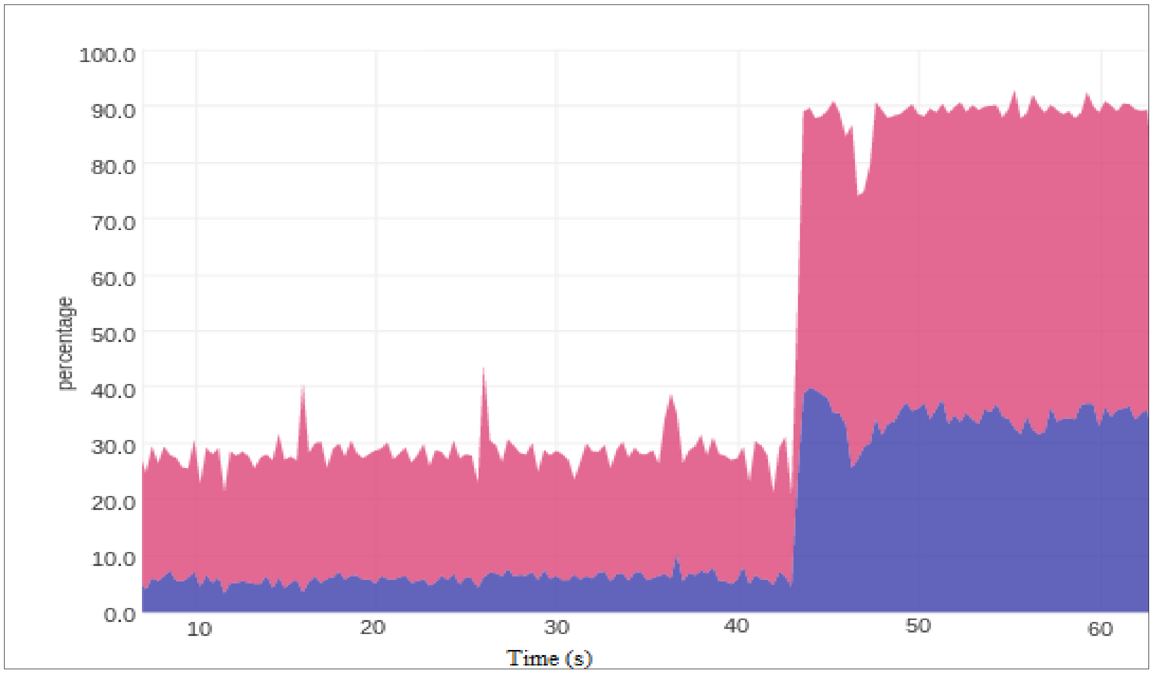


Figure 3 Control Plane CPU Utilization under UDP Saturation Attack

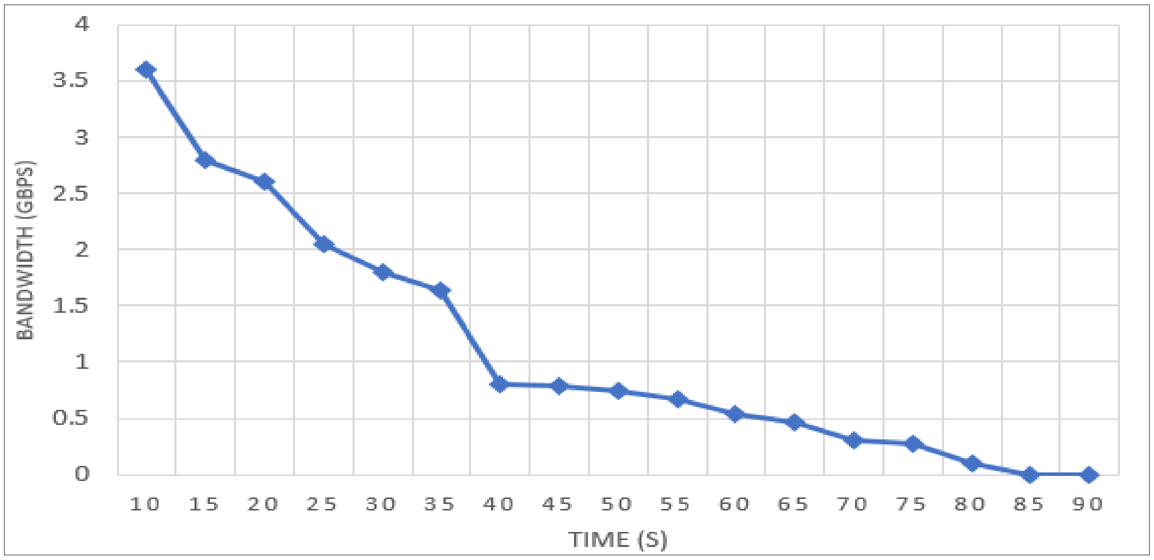


Figure 4 OpenFlow Connection Channel Utilization under UDP Saturation Attack

1.3 Problem Statement

The OpenFlow protocol provides a reactive packet processing approach which makes the SDN network more adaptable and agile to requirement changes of the network applications. An OpenFlow switch processes the packets by matching them with the installed flow-rules on the flow tables. When no flow-rules match the incoming packet (i.e., table-miss), the OpenFlow switch encapsulates this packet inside a Packet-In message and sends it to the controller to determine the proper action. After that, the controller computes the proper action and installs new flow-rules on the OpenFlow switch.

This reactive packet processing exposes a security vulnerability that can be exploited by an attacker to launch the data-to-control plane and control-to-data plane saturation attacks against the SDN infrastructure. An attacker can launch a data-to-control plane saturation attack by generating a huge number of table-miss packets, by sending a vast number of spoofed packets to reduce the possibility of matching any of the existing flow-entries on the victim switch. Thus, a large number of Packet-In messages forwarded to the controller will consume the computation resources of the control plane (data-to-control plane). Subsequently, a huge number of fake flow entries will be forwarded by the controller and exhaust the flow-table memory buffer of the victim OpenFlow switch (control-to-data plane). In the end, the entire SDN network will be paralyzed.

In protecting the SDN infrastructure from saturation attacks, we encounter the following issues:

- How to effectively detect the known and unknown saturation attacks which may compromise the SDN infrastructure?
- How to effectively identify the OpenFlow switches which are targeted by known and unknown saturation attacks?
- How to effectively countermeasure the malicious OpenFlow traffic without sacrificing the SDN network normal traffic, and how to effectively eliminate the saturation attack consequences on the data plane without losing the pre-attack settings?

These three problems are hard issues to overcome. For the first issue, a simple solution is to develop a detection method that starts periodically based on an arbitrarily pre-defined time window. However, if the time window is too long, the saturation attacks will take over the entire network, and if the time window is too short, the amount of traffic may be insufficient to have reliable results. In the latter case, the detection method will start more frequently, which may cause a performance overhead for the SDN environment. Thus, we should discover the proper time window, as well as the proper machine-learning classifier to detect the known and unknown saturation attacks with the highest detection performance.

For the second issue, we can simply inspect the behavior of all the OpenFlow switches of the SDN environment, one by one. However, this process causes a large performance overhead on the SDN environment and requires a long processing time. Thus, the saturation attack may take over the entire network. Therefore, we should provide an efficient method that can accurately identify the victim OpenFlow switches by known and unknown saturation attacks.

For the third issue, we should mitigate a family of saturation and hybrid saturation attacks without sacrificing the normal OpenFlow traffic, modifying the SDN/OpenFlow architecture, and/or adding extra devices. A simple solution to countermeasure these attacks is to install blocking flow rules that drop the table-miss packets. At this point, the legitimate table-miss packets from benign OpenFlow switches will be dropped. Thus, the benign OpenFlow switches will not be able to process all new incoming traffic flows generated from benign hosts. However, an appropriate solution should be able to handle the table-miss packets – specifically, the malicious table-miss packets – effectively and keep forwarding the normal traffic. Thus, it should distinguish the malicious OpenFlow traffic from the benign traffic, along with the zombie hosts and the targeted hosts, without the need for an extra device or modifying the SDN architecture (which is another challenging issue). Also, we should identify the fake flow-rules that have been installed on the victim switch flow-tables during the attack and remove them to enable the installation of legitimate flow-rules on the victim switches.

This dissertation introduces an SDN defense framework that can protect the control plane, data plane, and the OpenFlow connection channel against known and unknown saturation attacks without the need for any additional hardware or modifying the design of the SDN architecture. This makes the proposed defense system easily deployable in the current SDN environments. It can detect known and unknown saturation attacks by adopting the saturation attack detection module. Also, it can identify the targeted OpenFlow switches by using the victim switch detection module. Besides, it is capable of identifying the zombie hosts and the targeted destinations by using the Packet-In deep inspection filter. Furthermore, the proposed defense system can mitigate the saturation attacks by blocking the incoming malicious traffic from the zombie hosts and can remove the consequences of these attacks by eliminating the installed malicious flow-rules on the victim OpenFlow switches.

1.4 The Proposed Approach

The proposed approach protects the SDN network against saturation attacks. It can detect known and unknown saturation attacks, identifying the targeted OpenFlow switches by these attacks, and mitigating these attacks by blocking the malicious incoming traffic and eliminating their consequences.

Figure 5 shows the system architecture of the proposed approach. It consists of four modules: network topology manager, OpenFlow traffic collector and feature extractor, saturation attack detection, victim switch detection, and countermeasure.

Initially, the network topology manager module extracts the SDN environment topology by using the northbound REST APIs of the controller and classifies the extracted network topology based on the connected OpenFlow switches. Next, the OpenFlow traffic collector and feature extractor module will be triggered. This module is a session-based process that collects the OpenFlow traffic by incorporating the Pyshark library. For each session, the feature extractor extracts the saturation attack detection module features. The duration of each session is equal to the pre-defined time window of OpenFlow traffic analysis.

Before developing our online saturation attack detection module, we conducted extensive offline experiments using both physical and simulated SDN environments. These experiments allowed us to evaluate the detection performance of the supervised and semi-supervised algorithms by generating datasets from different time windows of OpenFlow traffic analysis. Based on the reported experimental results, we have been able to obtain the proper time window of OpenFlow analysis along with the highest-performing machine learning algorithms (on the known and unknown saturation attacks). In our approach, the OpenFlow traffic time window should equal 1 minute based on our findings, (see Chapter 3).

Upon extracting the detection module features, the saturation attack detection module will be triggered. This module is an anomaly detection module that is responsible for detecting the saturation attacks against the SDN network by incorporating the Variational Autoencoder algorithm.

When the detection module detects an abnormal behavior in the SDN environment, the victim switch detection module will be activated, and the OpenFlow traffic collector feature extractor module will extract its detection features. This module is an anomaly detection method that is responsible for identifying the targeted OpenFlow switches by known and unknown saturation attacks. It adopts the Variational Autoencoder algorithm (see Chapter 4).

When the victim switch module identifies the targeted OpenFlow switches, the countermeasure module will be activated for attack mitigation in the following three steps:

1. The Packet-In deep inspection component extracts the Packet-In messages of the identified OpenFlow switches by the victim switch detection module from the collected OpenFlow traffic and classifies them based on the OpenFlow switch. Next, it inspects the header of the table-miss packet inside the Packet-In messages to extract the zombie hosts and reduce the false-positive rate of the victim switch detection module.
2. The blocking flow-rule manager receives the list of zombie hosts with corresponding OpenFlow switches and installs high priority blocking rules.
3. The flow-table manager obtains the victim switch flow rules, extracts, and deletes the fake ones by using the attack topology.

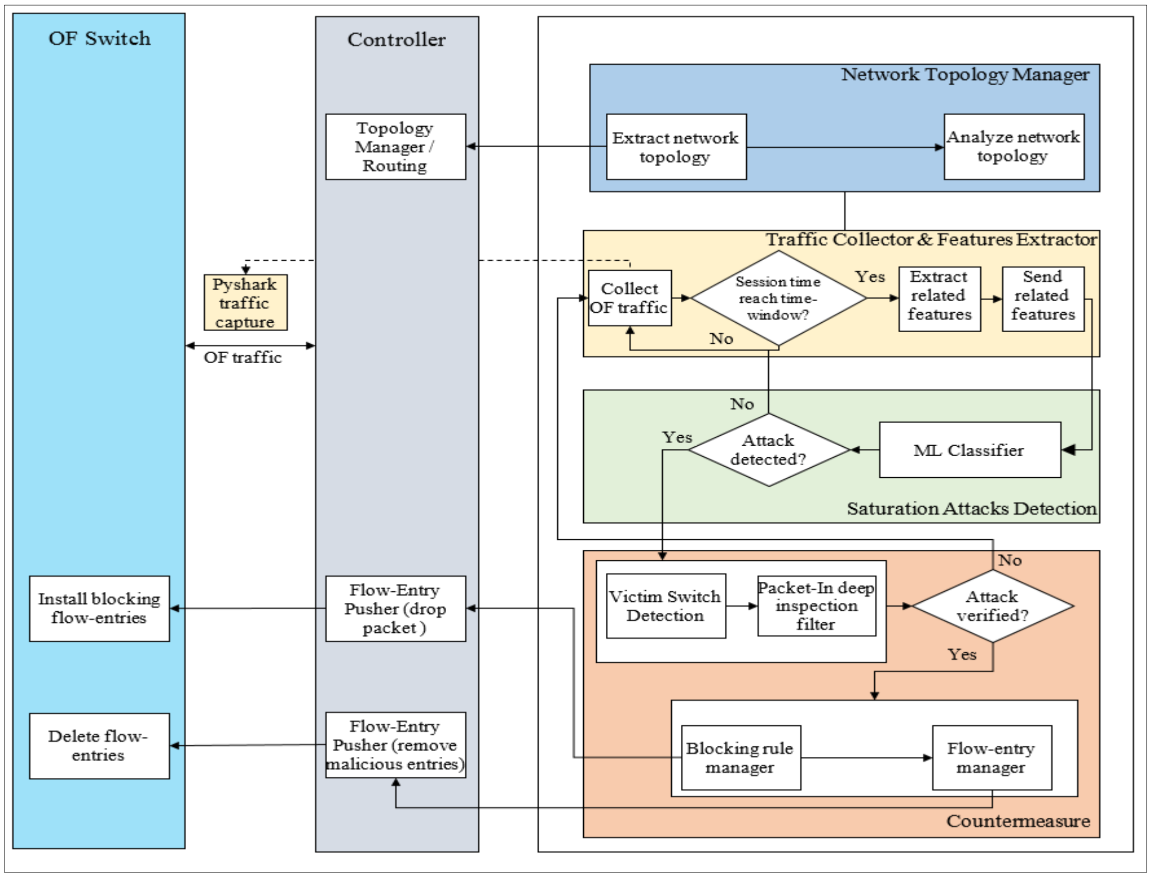


Figure 5 System Architecture

1.5 The Contribution

Recent state-of-the-art SDN defense systems focused on mitigating the TCP-SYN flooding attack against an SDN network. However, many of the proposed anomaly defense systems are deficient in detecting and mitigating a broader set of saturation attacks, as well as unknown saturation attacks. Also, the recent systems require modifications to the SDN/OpenFlow architecture, adding extra hardware, and/or extensive computational resources. This dissertation proposes an anomaly defense framework for SDN that can protect the control plane, data plane, and OpenFlow connection channel against known and unknown saturation attacks, without the need for any additional hardware or modification of the SDN architecture design.

The proposed defense system can detect SDN network saturation attacks by using the saturation attack detection module. Besides, it identifies the victim OpenFlow switches being targeted by known and unknown saturation attacks by utilizing the victim switch detection module. It also identifies the zombie hosts and the targeted destinations by using the Packet-In deep inspection filter. Finally, it can mitigate the saturation attacks by blocking the incoming malicious traffic and removing the consequences of these attacks (i.e., the installed fake flow-rules) on the victim OpenFlow switches.

The contribution of this dissertation is as follows:

- To the best of our knowledge, this work is among the first to investigate the impact of different time-windows of OpenFlow traffic on the performance of supervised classifiers for the detection of saturation attacks in SDNs.
- We present an in-depth study for different victim switch detection methods with the integration of supervised and semi-supervised machine learning

algorithms and provide an anomaly victim switch detection module that is capable of detecting and identifying the OpenFlow switches targeted by known and unknown saturation attacks.

- We provide a cost-effective countermeasure module that can defend the SDN network against a family of saturation attacks and remove their consequences.
- We design and implement the proposed defense system and evaluate it in comparison with the most recent effective defending systems, using extensive experiments that simulated the real-life SDN network. The reported results show that the proposed framework is an effective defense system for an SDN network, capable of detecting and mitigating the saturation attacks in real-time with very minimal resource consumption.

1.6 Dissertation Organization

The rest of this dissertation is structured as follows. Chapter two reviews related work. Chapter three introduces the SDN saturation attack detection module responsible for determining whether the SDN environment is being targeted by a saturation attack or not. Chapter four illustrates the victim switch detection module responsible for identifying the OpenFlow switches targeted by saturation attacks. Chapter five describes the countermeasure module that mitigates the saturation attack(s) and illustrates the implementation and evaluation of the proposed defense system. Finally, chapter six concludes the dissertation with a description of future work.

CHAPTER TWO: RELATED WORKS

2.1 Detecting Denial of Service (DoS) Attacks in Computer Networks

Denial of Service (DoS) attacks impose a security threat to all types of computer networks since DoS attacks can be easily launched and hard to detect. For example, in a traditional network, an attacker can launch a saturation attack by sending a large number of packets toward the network switches, with or without spoofing the source IP addresses. At this point, the network switches will be overwhelmed by processing the attack packets. In SDN, the network architecture is divided into multiple layers (i.e., application layer, control layer, and data plane) and all of these layers can be targeted by DoS attacks. For instance, an attacker can target the data plane layer by launching different kinds of saturation attacks, such as the UDP flooding attack, by sending a large number of spoofed packets in order to urge the targeted OpenFlow switch to generate the Packet-In messages. This, in turn, overwhelms the controller and saturates the OpenFlow connection channel.

In recent decades, different research works have been proposed to detect and prevent DoS attacks against traditional networks [3,4]. This section introduces these works to investigate the applicability of the proposed methods for detecting saturation attacks against an SDN environment.

To start, different research works have studied the adoption of supervised machine-learning classifiers to detect DoS attacks in traditional networks. Singh et al. [5] proposed an application layer DoS attacks detection method by adopting a genetic machine learning classifier (MLP-GA). They extracted features from the network traffic, such as the entropy value of the number of GET requests per connection, the entropy value of the IP addresses of GET requests', and the entropy value of the GET requests' counts. If the entropy values of the extracted features are high, this indicates that the network is under an application layer DoS attack. Fouladi et al. [6] proposed a detection method for DoS attacks by using Naïve Bayes classifiers. This approach used a Discrete Fourier transform (DFT) and discrete wavelet transform (DWT) as features. Mafra et al. [7] proposed the Octopus-IIDS intrusion detection system by incorporating the Kohonen Network and a Support Vector Machine (SVM). It consists of: (1) a classifier layer that uses the SVM to classify the data into four categories (i.e., DoS, U2R, probe, and R2L), and (2) an anomaly detection layer that incorporates the Kohonen Network to detect the anomalies based on the data classification of the previous layer. The system can be used on small scale networks.

Wagner et al. [8] proposed a DoS detection method using the one-class SVM classifier in order to detect new attacks. The detection method used the One-Class SVM algorithm. For such a classifier, the training dataset includes one class of traffic types (e.g., normal traffic or attack traffic). Based on the reported results, the accuracy of the proposed detection method is equal to 92% for all types of attacks. Muda et al. [9] introduced a two-stage DoS detection method. In the first stage, the K-means clustering machine learning algorithms were used to classify the collected traffic into three groups: (1) Prob, R2L, and U2R attack data, (2) DoS attack data, and (3) normal data. In the second stage, the Naïve Bayes classifier is used to classify the collected traffic into one of the aforementioned groups. Based on the reported results, the two-stage detection method is an effective one in detecting DoS and other attacks. However, due to the adoption of supervised classifiers, these detection methods [5-9] cannot be used to detect the unknown saturation attacks against SDNs. Besides, the features of these methods cannot accurately reflect the impact of saturation attacks in SDN.

Other detection methods adopt unsupervised machine learning algorithms to detect DoS attacks. For instance, Hsieh and Chan [10] used an Artificial Neural Network (ANN) to detect DoS attacks. They extracted several features from the network traffic packet headers: source IP address, destination IP address, the time of the packet, and length of the packet. In addition, they used Apache Spark to process the extracted features from the network traffic and convert them into a vector suitable for the ANN model. Bhuyan et al. [11] proposed an anomaly detection method by integrating an unsupervised machine learning algorithm for large datasets. It uses the tree-based subspace clustering to obtain a high detection rate. The reported results showed that the detection method achieved 98% accuracy.

Yuan et al. [12] introduced DeepDefense, a DoS attack detection method that integrates a Recurrent Neural Network (RNN). They used the UNB ISCX Intrusion Detection 2012 dataset to train and evaluate the RNN model. This detection method obtained high detection results with 97% accuracy and 97% recall.

Yadav and Subramanian [13] proposed an application-layer DoS attack detection method using the Stacked Autoencoder deep learning algorithm. They used 10 features to train the model and they created a testing dataset in order to evaluate the proposed approach. Qin et al. [14] presented a DoS attack detection method based on entropy clustering. This approach can be divided into two phases. First, a clustering phase extracts the entropy values of the source and destination IP addresses, port numbers, packet size, and duration. They use a clustering algorithm to build many clusters based on the extracted entropy values. Second, the method consists of a detection phase in which they detect the DoS attacks by using the distance between the clusters.

Saad et al. [15] proposed the v6IIDS framework to detect ICMPv6 saturation attacks by adopting the Artificial Neural Network Back-Propagation algorithm. The proposed defense system consists of: (1) a data collection and preprocessing module that collects the network traffic and extracts features for the detection module, and (2) an anomaly detection method that is responsible for detecting the ICMPv6 attack by using the ANN back-propagation trained model. Yin et al. [16] proposed a detection method using a Recurrent Neural Network (RNN) to detect DoS attacks. In this approach, they relied on the NSL-KDD dataset with the NSL-KDD dataset 41 features.

Farnaaz and Jabbar [17] proposed a DoS detection method using the Isolation Forest algorithm. They also used the NSL-KDD dataset to train their isolation forest model. Based on the reported results, the trained model obtained 99.2% accuracy. Malik et al. [18] proposed a DoS attack detection approach utilizing Particle Swarm Optimization (PSO) and Random Forest (RF) algorithms. The proposed method can be divided into two phases: (1) a feature selection phase, which adapts PSO techniques to select the most appropriate features from the KDD99 Cup dataset, and (2) a detection phase, which the RF classifier uses to detect the DoS attacks. However, most of the anomaly detection methods [10,11,12,14,15,17,18] relied on an outdated dataset that did not represent the nature of the SDN traffic. Also, the extracted features such as “time-zone and port number” cannot express the behavior of saturation attacks against SDNs, specifically, in a reactive packet processing SDN environment. Thus, the adoption of such detection methods would leave the SDN environment vulnerable to the majority of saturation attacks.

Different detection methods have employed statistical algorithms to detect DoS attacks in the traditional network. For example, David and Thomas [19] used a fast entropy value to detect DoS attacks. They calculated the fast entropy of flow-count and compared it against a predefined threshold to detect the DoS attack. A low fast entropy value means there is a DoS attack and a high entropy value means no attack, since the attack flow is dominant over other normal flows.

Hoque et al. [20] presented a detection approach using the Multivariate Correlation Analysis (MCA) algorithm. The working process of the proposed approach is (1) to collect the network traffic and divide it into multiple time windows, (2) to calculate the packet rate and the entropy values and variational index of the source IPs, and (3) to use MCA to find the correlation between the extracted features. Subsequently, they compare the extracted deviations against a predefined threshold to detect the DoS attacks. The predefined threshold incorporates multiple assumptions – for example, the assumption that a high entropy value of source IPs with a high entropy value of packet rates, means that the possibility of a DoS attack is high. However, in the reactive SDN environment, the features of the proposed detection methods [19, 20] would need to be tuned to calculate the entropy value of the table-miss packet rather than the traffic packets. Otherwise, the proposed methods could produce a large number of false alarms.

2.2 Detecting Saturation Attacks in SDN

As a new network paradigm that provides agility and programmability, SDN attracts industry and academia researches worldwide. Different research studies have shown various security threats in SDN [21, 22]. For example, DoS network flooding attacks disturb the SDN-based network and render it out-of-service. Different research works have been proposed to detect, mitigate, and prevent the SYN-Flooding attack by using various machine learning and deep learning approaches [23].

Asharf and Latif [24] discussed the possibility of adopting machine learning approaches in SDN to detect the DoS attacks. However, this work did not go so far as to investigate potential approaches that can be used to detect OpenFlow switches targeted by known and unknown saturation attacks. Niyaz et al. [25] proposed an SDN network application that adapted the Stack Autoencoder (SAE) deep learning technique for detecting multi-vector DDoS. The proposed defense system consists of three components: Traffic Collector and Flow Installer, Feature Extractor, and Traffic Classifier. This work relies on processing every incoming packet for attack detection and flow computation, which requires extensive computational resources, instead of flow sampling. Furthermore, the dataset that was used for training and testing the proposed defense system was collected from a traditional wireless network, which is not an SDN-based network.

Aizuddin et al. [26], proposed a DDoS detection prototype by using a Dirichlet Process Mixture model to detect the attack traffic. With this system, the misclassification rate of the attack traffic is around 50%.

Braga et al. [27] adopted the self-organized map (SOM) to develop a lightweight detection system for DDoS flooding attacks against SDNs. The proposed defense system consists of three modules: a flow collector module that collects all the flow entries of the connected OpenFlow switches, a feature extractor module that extracts the detection module features from the collected flow-entries, and a SOM classifier module that detects the attack traffic. However, this work requires extensive processing time to extract the detection module features, since it needs to process all the flow entries of connected

OpenFlow switches. This delay may give the attacker enough time to flood the whole environment. Also, when an OpenFlow switch is targeted by a saturation attack, it becomes overwhelmed from processing the malicious table-miss packets. Thus, the flow collector module will not obtain the flow-rules of the targeted OpenFlow switch in real-time. In addition, the flow-rules' messages are large-size messages, which may help in saturating the connection OpenFlow channel between the controller and the OpenFlow switches.

Tang et al. [28] used the Deep Neural Network (DNN) to develop an anomaly DoS detection system. The accuracy of the proposed detection model is relatively low – just 88.04%. Also, the NSL-KDD dataset used in training and testing the detection model was generated from a traditional network.

Abubakar and Pranggono [29] developed a flow-based anomaly detection system by using a neural network. Again, the NSL-KDD dataset was used to train and evaluate the models, which is the main shortcoming of this approach.

Mousavi and Hilaire [30] proposed an early DoS attack detection method by calculating the entropy values of the IP addresses of the first 250 packets forwarded to the controller. This approach assumes that each new packet forwarded to the controller is a malicious packet if the destination address matches any of the already-existing network hosts. Also, if the destination IPv4 address appears in many packets, the entropy value will be lower than the predefined thresholds, and the system will think that an attack is occurring. This approach can generate many false alarms of early attack detection, specifically, if the SDN network is in a reactive flow-management configuration. In this type of configuration, the flow-entries are configured dynamically to provide a flexible way to control the network traffic. Thus, in a large-size SDN network, many legitimate new packets forwarded to the controller and many legitimate flow-entries will be installed to control network traffic. However, the proposed approach cannot be utilized as an early DDoS attack detection method since the normal behavior of the SDN network will always be considered malicious behavior.

Azizz and Okamura [31] proposed the FlowIDS framework to detect Simple Mail Transfer Protocol (SMTP) flooding attacks by using decision tree (DT) and deep learning (DL) algorithms to detect the malicious SMTP flow traffic. The deep learning algorithm and the decision tree classifier were trained to identify the benign SMTP traffic. Subsequently, both the DL and DT were used to detect the attack SMTP traffic. However, this work cannot be used to protect the SDN environment against saturation attacks. Also, using two machine learning algorithms at the same time to identify the attack SMTP traffic may require a long prediction time. Besides, the authors did not provide more details when the two algorithms obtained different prediction results.

Santos et al. [32] introduced the ATLANTIC framework to detect DDoS attacks against the SDN environment. The proposed defense system consists of two phases: (1) a lightweight processing phase that can be executed periodically to detect the deviations of the SDN network traffic flows by using entropy analysis in order to identify the suspicious traffic flows, and (2) a heavyweight processing phase that uses a K-means unsupervised algorithm to cluster the similar traffic flows and then adopts an SVM classifier to classify the malicious flows from the normal ones. The ATLANTIC framework has three main components: (1) a statistical layer that is responsible for collecting the traffic flows statistics, (2) a classification layer that is responsible for detecting and classifying the malicious traffic flows and, (3) a network layer that is responsible for tracking the SDN data plane and collecting the traffic flow information from the controller. The detection performance of the proposed defense system is relatively low; it obtained 88.7% accuracy and 82.3% precision. Also, this work caused performance overhead on the SDN environment due to the processing time and it required a long prediction time due to using the SVM and K-means classifiers together. Besides, the proposed defense system cannot detect unknown saturation attacks and countermeasure them.

Ye et al. [33] proposed a detection system for UDP, SYN, and ICMP flooding attacks by using an SVM classifier. The proposed defense system includes: (1) a flow state collection module that collects the status of the OpenFlow switches flow-tables by using controller-to-switch messages, (2) Characteristic Values Extraction module that is responsible for extracting the classifier features (it extracts 6 features from the collected flow tables' status messages), and (3) classifier judgment module, which utilizes an SVM classifier to detect the attacks. This work represents a simple method to detect some flooding attacks in SDNs. However, using the controller-to-switch messages to collect the flow-table status information can cause performance overhead, specifically, when the environment is under attack. This is because the size of the flow-tables status messages is large, which may help to saturate the OpenFlow connection channel.

Lee et al. [34] introduced the Athena framework, which exposed different APIs and allowed researchers and developers to easily integrate their anomaly detection applications with SDN environments. The main goal of this research is to highlight the problem of integrating different detection systems into SDN deployment. Athena offers high-level APIs called Athena NB interfaces, which enable the developers to develop anomaly detection methods, and Athena SB interfaces, which isolate the complexity of dealing with the SDN data plane.

Chen et al. [35] proposed a detection method for DNS and TNP reflection amplification attacks. The detection method consists of a detection agent module. The detection agent module contains (1) a traffic collector that uses the Netmate tool to collect the traffic and calculate the features vector, and (2) an SVM-based machine learning classifier that detects the DNS and NTP reflection attacks using the extracted features vector. This research is dedicated to detecting two attacks by using an SVM classifier. However, this detection method cannot detect major saturation attacks, such as the UDP flooding attack, or unknown saturation attacks.

Alshamrani et al. [36] introduced two new attacks, the Misbehavior and NewFlow attacks, and proposed a detection method that is capable of detecting these new attacks and other DDoS attacks. The Misbehavior attack is a kind of attack that disguises the first packet of the forwarding flows like a normal packet, while the remaining packets of the flow are malicious ones. The NewFlow attack is the same attack as a data-to-control plane attack. The proposed system adopted a Sequential Minimal Optimization (SMO) classifier to detect these attacks and used the NSL-Dataset to train the SMO classifier. However, the proposed new attacks (i.e., Misbehavior and NewFlow attacks) are not new ones since all the saturation attacks behave in the same way. Also, the NSL-Dataset is not an SDN dataset, which may raise a question about the feasibility of using the proposed detection method in real-life SDN environments.

Wang et al. [37] proposed a scalable method to detect the TCP-SYN attack in an SDN environment. The detection method collects the numbers of SYN and FIN packets during a period of time and provides these numbers to the Change Point Detection statistical algorithm to find out the homogeneity between the numbers of SYN and FIN packets. If there is any heterogeneity found between these numbers at some point of time, a TCP-SYN attack is detected. This method cannot be used to detect known and unknown saturation attacks in an SDN environment.

2.3 Detecting Victim OpenFlow Switches in SDN

Identifying the targeted OpenFlow switches did not garner much attention in academic research. Recently, a few studies have been proposed to tackle this issue. Po-Wen et al. [38] proposed a simple detection method that samples flow-rules from randomly selected OpenFlow switches. Next, they generate artificial packets to see if the OpenFlow switch executes the corresponding flow-rules correctly. This approach may produce a high rate of false-positives since the flow-rules of the OpenFlow switches are changing over time.

Zhou et al. [39] proposed SDN-RDCD, a real-time approach to detect the targeted SDN devices when the controller and OpenFlow switches are not trustworthy. SDN-RDCD uses a backup controller as an audit controller that is responsible for recording the network update events information such as deleting, adding, or updating flow rules from the original controller and its connected OpenFlow switches. Subsequently, the audit controller allocates a unique audit ID for each update request event and records it in an audit record. This audit ID is used to keep track of each event, as well as the execution results on the original controller and corresponding OpenFlow switches. Also, the audit ID is used by the audit controller to re-execute the update event and record the execution results. Then, SDN-RDCD analyzes the recorded audit log to extract any inconsistency of the handling of information by the controller and OpenFlow switches. However, this work may require a long time to process the audit records in order to find the unmatched event handling information. Thus, the saturation attacks may compromise the entire-network before detecting the victim OpenFlow switches. Also, this approach cannot detect most of the OpenFlow switches that were targeted by saturation attacks, since the behavior of these victim switches is very similar to the normal ones. In addition, this approach cannot be easily adopted in real-life since it requires adding an extra controller as an auditor-controller.

Different from the aforementioned works, the victim switch detection method proposed in this dissertation, is an effective method that is capable of detecting the OpenFlow switches that are targeted by known and unknown saturation attacks. Also, it can be easily deployed in real-life SDN environments since it does not require any modification of these environments' architecture.

2.4 A Countermeasure to Saturation Attacks in SDN

Different studies have been proposed to defend the SDN against saturation attacks. For example, Hu et al. [40] introduced the FDAM system for detecting UPD, ICMP, and SYN flooding attacks. It consists of two modules: (1) an attack detection module that is responsible for detecting DoS attacks by using an SVM classifier and a sFlow approach to collect the network traffic and extract features, and (2) a DoS attacks mitigation module that mitigates flooding attacks by using traffic migration and white-list approaches. Unfortunately, the SVM classifier requires a long training and prediction time. Also, based on our reported results in this research, the SVM classifier is not capable of detecting the unknown saturation attacks.

Seungwon et al. [41] proposed the AVANT-GUARD framework to mitigate the TCP-SYN flooding that is sent to the SDN controller. It accomplishes this task by extending the OpenFlow-Switches functions. The detection module monitors the ongoing TCP-SYN connections to the controller and detects the SYN flooding based on a predefined threshold, which cannot accurately differentiate between the normal and abnormal SYN packets.

Wang et al. [42] proposed FloodGuard as a prevention approach against DoS attacks. FloodGaurd acts as middleware between the controller and its applications and has three components: a detection module, a flow rule analyzer module, and a packet migration module. The FloodGaurd detection module monitors the OFPT_PACKET_IN messages and triggers the other modules when the OFPT_PACKET_IN messages exceed the pre-defined thresholds.

Shang et al. [43] proposed FloodDefender as an SDN application to protect the control plane and data plane against DoS attacks. It has four modules: attack detection to detect the DoS attacks, table-miss engineering to migrate the table-miss packets to the neighbors' switches, packet filtering to identify the attack traffic, and flow rule management to remove the useless flow-rules. The main limitations of this work are as follows: (1) All the table-miss packets are delivered to the controller to process them, whether they are normal ones or not. (2) There is a relatively high flow-table utilization due to the installation of protecting and monitoring flow-rules. (3) The attack detection module uses the Packet-In messages rate to detect the attack, which is not an effective approach since a high volume of new normal traffic can produce similar Packet-In messages. (4) FloodDefender cannot be applied in a small-scale SDN network. Lastly, (5) if two or more of the switches are regarded as victims, this approach will not be able to handle the saturation attacks and may flood the whole SDN network.

Menghao et al. [44] introduced two novel attacks – a table-miss striking attack and a counter manipulation attack – and provided the SWGuard system as a solution to detect and prevent these attacks.

Yan and Huang [45] proposed a DDoS detection and mitigation system (DDMF) that detects and mitigates the impact of DDoS attacks in real-time by adopting SDN features and Apache Spark. DDMF consists of three components. (1) a capture server is responsible for collecting the network traffic and saving it in a log file by using Apache Spark. (2) a detection module incorporates a neural network (NN) to detect the attacks based on the integrity of the log file. The detection module is installed on a detection server. (3) an SDN router application is responsible for mitigating the attacks by sending the traffic to the cleaning centers. This approach cannot be easily deployed into SDN environments since it requires a detection server and cleaning centers to analyze and mitigate the attack traffic.

Jing et al. [46] proposed the FL-Guard detection and defense system against DDoS attacks in the SDN environment. FL-Guard is implemented as an SDN application that resides in the application layer of the Floodlight controller. FL-Guard uses sFlow-RT to collect network traffic. FL-Guard includes two modules: (1) an attack detection module that incorporates an SVM classifier to detect the DDoS attacks, and (2) an attack-blocking module that blocks the attack from the source port. The shortcomings of this work are as follows: (1) sFlow-RT uses a periodical sampling of the OpenFlow traffic and cannot collect information on all OpenFlow packets. This may cause a large impact on the accuracy of the detection method. Also, sFlow-RT cannot collect a low-rate of OpenFlow traffic. (2) The SVM classifier cannot detect unknown saturation attacks since they are not included in the training phase. Thus, if the SDN environment is targeted by an unknown saturation attack, the detection module will not be able to detect it and the blocking module will not be able to mitigate this attack.

Cui et al. [47] proposed the SDN-Anti-DDoS system, which is capable of detecting DDoS attacks quickly. The proposed system consists of four modules. (1) The attack detection trigger module monitors and counts the velocity of Packet-In messages to detect the abnormal burst of Packet-In messages. It uses the exact-Storm machine learning algorithm and triggers the other modules if the SDN environment is under attack. (2) The attack detection module adopts a Neural Network (NN) algorithm to distinguish the malicious flow-entries from normal ones. This module uses controller-to-switch messages to identify the flow-entries and extract features for the NN model. (3) The attack traceback module uses the same NN model to obtain the attack information. Finally, (4) the attack mitigation module mitigates the attacks by installing blocking entries. This work may cause a high amount of false alarms, specifically, in a reactive processing SDN environment when a burst of normal traffic is forwarded by legitimate applications.

Also, using control-to-switch messages to obtain the switches' flow-entries when the SDN environment is under an attack makes it harder to detect the attacks in real-time, because the targeted OpenFlow switches will be overwhelmed by malicious Packet-In messages and the controller will be busy processing the forwarded Packet-In messages. Thus, the detection module will not obtain the features required to detect the attack in a timely manner, which may give the needed time to the attacker to destroy the entire network.

Besides, the flow-entries messages are large in size, which may help to saturate the OpenFlow connection channel.

Durner et al. [48] proposed a detection and mitigation method for overflow attacks against the SDN data plane. The detection method adopted a statistical method to detect the attacks by using the flow-tables' header fields. It keeps a table of the suspected table-headers using hashing. Based on that table, the mitigation method blocks the attacks. This work causes a large number of false-positives, since the flow-rules of the OpenFlow switches are changed frequently. Also, the detection and mitigation method cannot be used against known and unknown saturation attacks.

Reza et al. [49] introduced SLICOTS as an SDN defense system against TCP-SYN flooding attacks. The proposed system monitors all TCP handshaking processes between the SDN hosts to install temporary forwarding rules on the OpenFlow switches during the handshaking process. If the half-open TCP connections between a host and a server exceed the predefined threshold, a TCP-SYN flooding attack is detected. Subsequently, it installs blocking rules to stop the malicious SYN packets.

This system is capable of protecting the control plane from the TCP-SYN attack and ignores the protection of the data plane. Also, this approach installs temporary forwarding rules on all connected OpenFlow switches, without determining which ones are targeted, in order to count the SYN connections.

This procedure may exhaust the data plane memory. In addition, the proposed detection and prevention method cannot be used to protect the SDN environment against known and unknown saturation attacks. Therefore, the SDN environment is vulnerable to the majority of saturation attacks.

Ficherta et al. [50] proposed the OPERETTA, an OpenFlow defense system against TCP-SYN flooding attacks. OPERETTA has been implemented as a controller application that can detect fake TCP-SYN connections and reject them. Similar to SLICOTS [49], OPERETTA counts the number of TCP-SYN connections and matches them against a predefined threshold. If the counter value exceeds the predefined threshold, a TCP-SYN flooding attack is detected. This system is deficient in protecting the SDN environment against known and unknown saturation attacks.

Different from the aforementioned works, the proposed defense system in this dissertation can detect the known and unknown saturation attacks, as well as the OpenFlow switches that are targeted by these attacks. We have studied different victim detection methods by incorporating supervised and semi-supervised algorithms to identify the most effective method and algorithm. In addition, instead of using controller-to-switch messages or the sFlow-RT tool, we adopted the Pyshark library to collect the OpenFlow traffic in real-time in order to eliminate any performance overhead on the controller, OpenFlow switches, and OpenFlow connection channel. Also, the proposed defense system provides a countermeasure method that can effectively mitigate a family of these attacks without the need for adding extra hardware, modifying the SDN design, or causing performance overhead on the SDN environment. Besides, it can remove the fake flow-rules that have been installed on the victim OpenFlow switches during the attack, enabling the legitimate flow-rules to be installed.

CHAPTER THREE: DETECTION OF SATURATION ATTACKS

The problem studied in this chapter is how to detect saturation attacks in SDN environments by using state-of-the-art machine learning algorithms. The saturation attacks range in duration, and a long-lasting one of these attacks can affect the entire SDN environment. Therefore, to protect the computational resources of the SDN network, the proper solution is to detect these attacks at the early stages before they take-over the entire network. In the designing of the saturation attack detection module, there were two issues.

First, we should be able to obtain the proper time-window of OpenFlow traffic analysis as well as of the machine learning classifier to detect the saturation attacks with a high detection performance. So far, most of the existing machine learning detection methods rely on an arbitrary predefined, fixed time-window to start analyzing the network traffic to detect saturation attacks. However, if the time window is too large, the detection method response time will be long, and the attack may saturate the entire network. If the time window is too small, the amount of traffic may be inadequate to obtain accurate detection results. Also, the detection method will cause performance overhead over the SDN controller since it will be executed frequently. Thus, identifying the proper time-window for running the detection method and analyzing the traffic is a crucial point.

Secondly, machine learning approaches have deficiencies in detecting unknown saturation attacks. An unknown attack is an attack which is not represented in the dataset used to train the attack detection model [51]. Because there are no instances of the attack included in the training set, supervised machine-learning methods are unable to classify it.

Thus, evaluating the supervised and semi-supervised classifiers detection performance of unknown saturation attacks is another concern.

For the first issue, we have evaluated the detection performance of state-of-the-art machine learning algorithms, specifically, the widely-used Support Vector Machine (SVM) [52], K-Nearest Neighbor (K-NN) [53] classifier, and Naïve-Bayes (NB) [54] classifier. We used a variety of time-windows of OpenFlow traffic analysis to determine the proper time-window for the detection of saturation attacks as well as the most effective machine learning classifier. In addition, we studied the impact of different time-windows of OpenFlow traffic analysis on the machine learning classifiers' detection performance by conducting a false-negative analysis.

For the second issue, we evaluated the supervised machine classifiers such as SVM, K-NN, and NB classifiers and semi-supervised algorithms such as One-class SVM, Isolation Forest, Basic Autoencoder, and Variational Autoencoder performances for detecting unknown saturation attacks. In the experiments, we excluded the observations related to one type of saturation network attack from the training dataset, to act as an "unknown" attack for the models. The test dataset included the observations from the unknown attack and a random set of normal traffic observations.

Before implementing the proposed defense system, extensive experiments have been conducted using both physical and simulated SDN environments through offline settings. Therefore, the saturation attack detection method can be incorporated into the proposed defense system to detect the saturation attacks in online settings.

In the upcoming sections, we first introduce the features that were extracted from the OpenFlow traffic and explain the approach for preprocessing the OpenFlow traffic to generate multiple data sets for different time windows. We continue by discussing the supervised and semi-supervised algorithms used in this research for detecting the known and unknown saturation attacks and describing the experimental setups and OpenFlow traffic collection. Then the validation metrics used in this research to evaluate the performance of the machine learning classifiers are described, finally, the experimental results are presented.

3.1 Feature Extraction and Data Preprocessing

OpenFlow traffic is a sequence of packets that are transferred between the controller and the OpenFlow switch. Each packet has different attributes such as the packet time, the source and the destination IP addresses, the OpenFlow message type, and the length of the packet. Formally, the OpenFlow traffic, $OF,$ is a sequence of OpenFlow packets $< p_1, p_2 \ldots, p_\mathrm{n} >$ captured during a normal or attack session, where each packet, p_i, has $<time,\ srcIP,\ dstIP,\ OF\,msg,\ length>$.

OpenFlow traffic consists of 29 types of OpenFlow messages that can be categorized into three main types: (1) controller-to-switch messages that are sent from the controller to the switch to acquire information and modify the switch state (e.g., Packet-out, Packet-mod, and Role-request), (2) asynchronous messages that are sent by the switch to the controller to inform about new incoming packets, errors, and switch state changes (e.g., Packet-in, Flow-removed, Port-status, and error), and (3) symmetric messages that are sent between both sides such as Hello and Echo messages.

From the captured OpenFlow traffic, four features are extracted: 1) number of OFPT_PACKET_IN messages sent from the switch to the controller, (2) number of OFPT_PACKET_OUT messages sent from the controller to the switch, (3) number of

OFPT_PACKET_MOD messages sent from the controller to the switch, (4) number of TCP_ACK messages sent from the switch to the controller, or vice versa. These features are selected based on the analysis and observation of the OpenFlow traffic behavior in physical and in simulated SDN environments which are in attack mode and normal mode. The features are sensitive to saturation attacks as well as to hybrid saturation attacks, which are a combination of different saturation attacks that target the SDN environment.

The saturation and the hybrid saturation attacks have a different impact on these features. Figure 6 shows the impact of the UDP saturation attack on the Packet-in and Packet-out features. When the UDP saturation attack occurs, the zombie hosts try to flood the SDN network by generating a massive amount of IP packets including UDP datagrams. Meanwhile, the OFPT_PACKET_IN messages generated by the UDP attack come from the switch that is connected to the zombie's host. Thus, the number of OFPT_PACKET_IN messages in the OpenFlow traffic increases significantly, and the number of OFPT_PACKET_OUT messages decreases significantly. Table 1 summarizes the changes to each feature caused by UDP, SYN, ICMP, IP Spoofing, and SARFU TCP saturation attacks.

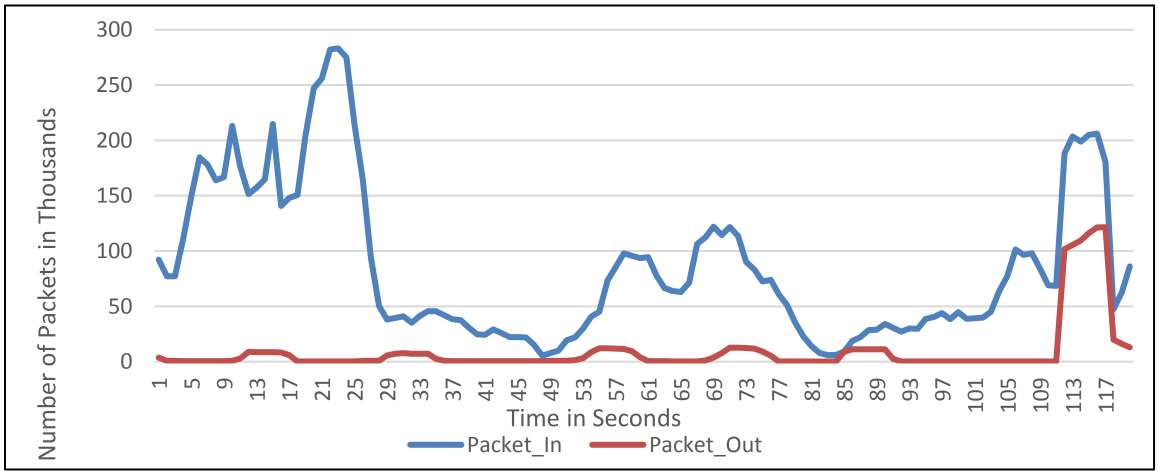


Figure 6 Effect of UDP Flooding Attack on Packet-In & Packet-Out Messages

Table 1 Impacts of Saturation Attacks on the Key OpenFlow Messages

Saturation Attack	**The Impact**			
	#Packet_In	**#Packet_Out**	**#Packet_Mod**	**#TCP_ACK**
UDP	Significant increase followed by a decrease	Significant decrease	Significant decrease	Significant increase
SYN	Significant increase	Increase and then a significant decrease	Increase and then a significant decrease	Significant increase
ICMP	Insignificant increase	Increase for short period of time followed by a significant decrease	Increase for a short period of time followed by a significant decrease	Noticeable increase followed by significant decrease
IP Spoofing	Increase followed by significant decrease	Increase followed by significant decrease	Increase followed by significant decrease	Increase and significant decrease to be a zero packet
SARFU TCP	Increase and then noticeable decrease and then decrease to zero packets.	Increase and then significant decrease to zero packets.	Increase and then noticeable decrease and then a significant decrease to zero packets.	Increase and then noticeable decrease.

To discover the appropriate time-window for detecting saturation attacks, we tested different time windows of OpenFlow traffic analysis and evaluated their impact on the detection performance of the SVM, K-NN, and NB classifiers. From each time window, a different dataset was generated from the collected OpenFlow traffic in both physical and simulated SDN environments. The dataset's time-windows ranged from one minute to the attack duration. A detailed description of our approach for extracting these datasets from the OpenFlow traffic is given in Algorithm 1. The features collected in each dataset were the OpenFlow traffic session OF, the dataset time-window T, the dataset time-shifting *S*, and the OpenFlow traffic type L, which is $< L \leftarrow 0\ if\ OF\ is\ normal\ or\ L \leftarrow 1\ if\ OF\ is\ attack >$. The output was a dataset X_J = $(\mathrm{x}_1, x_2, x_3 \ldots, x_\mathrm{n})$ which was a sequence of labeled samples, with each sample x_j in the form of <*number of Packet_in messages, number of Packet_out messages, number of Packet_mod messages, number of TCP_ACK message* >.

Lines (9-18) deal with extracting the features from the OpenFlow traffic packet sequence *OF* for the specified time-window *T*. For each packet, we extracted the OpenFlow message type. If the message type matched any of the messages, the corresponding counter increased by one (lines 10-17). When the difference between the packet time and the starting time is larger than *T*, a new sample x_j was created with the corresponding label and increased the dataset index J by one (lines 5-8). After each new sample x_j, the value of the $firstPacketIndex$ updated by adding the shifting parameter S for the next shift starting index (line 20). The reason behind including the shifting parameter was to increase the overlapping in the generated datasets.

Algorithm 1: OpenFlow Dataset Generation

Input	OpenFlow traffic OF, Time-Window T, Time-Shifting S, Traffic-Type L {0,1}
Output	Dataset X_J
Declare	packetIn, packetOut, packetMod, tcpAck, $Msg_{(type)}$
Steps	

```
1   J=0 (the index of the output sample $X_J$)
2   Repeat:
3     firstPacketIndex=1 ($p_i$ is the first packet of the current shift)
      $startTime = p_{time_{(firstPacketIndex)}}$ (is the first packet time of the current shift)
4      for ( $i =$ firstPacketIndex; $i < n$; $i++$) do
5         if $p_{time} - startTime > T$
6            createNewSample_Xj(packetIn,packetOut,tcpAck,J++)
             addNewSampleXj←xj with corresponding traffic type L
            and increase J by one.
7           break;
8         Endif
9          switch ($Msg_{(type)}$) {
10           Case1: $Msg_{(type)}$ = “OFPT_PACKET_IN”
11                packetIn += 1;
12            Case2: $Msg_{(type)}$ = “OFPT_PACKET_OUT”
13                  packetOut + = 1;
14           Case3: $Msg_{(type)}$ = “OFPT_PACKET_MOD”
15                   packetMod += 1;
16            Case4: $Msg_{(type)}$ = “TCP_ACK”
17                  tcpAck + = 1;
18         }
19     end for
20  firstPacketIndex=updatePacketIndexNextShfit(firstPacketIndex,S)
21  Until firstPacketIndex > n
```

3.2 Supervised and Semi-Supervised Classifiers

We studied the adoption of supervised and semi-supervised machine learning algorithms to determine the most accurate and effective approach for detecting known and unknown saturation attacks against SDN networks. Several supervised and semi-supervised classifiers have been trained using the datasets X_J that were obtained in the previous sections and evaluated their detection performance. In this work, K-NN, SVM, and NB classifiers were adopted as supervised machine learning classifiers. These classifiers have been widely used to detect DoS attacks in SDNs, since they are robust even with a noisy training dataset. However, these classifiers require a training dataset that includes a large number of specimens of all saturation attack types, which is hard to obtain in real-life. Therefore, these classifiers are deficient in detecting the OpenFlow switches which are targeted by unknown saturation attacks. The unknown saturation attack is an attack that has been mislabeled by the training model due to the absence of similar samples in the training dataset.

Therefore, the semi-supervised machine learning algorithms such as One-Class SVM [55], Isolation Forest [56], Basic Autoencoder [57], and Variational Autoencoder [58] were selected. These algorithms can be trained without the need to label the training dataset observations. Also, they can be trained using an out anomalies dataset (i.e., normal traffic dataset) and obtain a high anomaly detection result, such as the Variational Autoencoder algorithm.

Autoencoder is an artificial neural network composed of two functions: the encoder and the decoder. The encoder function E is a neural network transforming the original features x in a new space $y = E(x)$. Usually, y, is in a lower dimension of the original space. The decoder function D transforms the features Y from the new space to the original space $x^{\sim} = D(y)$. This neural network is trained by minimizing the reconstruction error, i.e. $loss = \left||x - x^{\sim}|\right|^{2}$. Once trained, the reconstruction error on a new example can be used as a score to detect whether an example is a training example or not. The smaller the reconstruction score, the higher the likelihood that the new example is similar to the one used in the training. For similar inputs vectors *x,x’,* a basic autoencoder can generate very different encoding representations *y,y’*. Thus, similar instances cannot be placed in the same encoded space, since it may lead to poor detection performance.

To overcome this issue, Variational Autoencoders are defined. A Variational Autoencoder is represented by a distribution of vectors, rather than a vector as the basic autoencoder, as shown in Figure 7. The encoder network generates the mean vector $\mu = \mathrm{E}_{\mu}(X)$ and the covariance matrix $\Sigma = E_{\Sigma}(X)$ that are used in a multivariate Normal distribution $\mathcal{N}(\mu, \Sigma)$ generating the encoding $\gamma \sim \mathcal{N}(\mu, \Sigma)$. In addition to the standard reconstruction error, the Variational Autoencoder imposes that the distribution of the encoded vector is approximately similar to a normal distribution with mean zero and standard deviation (i.e. they use the encoding Kullback-Leibler divergence regularization term).

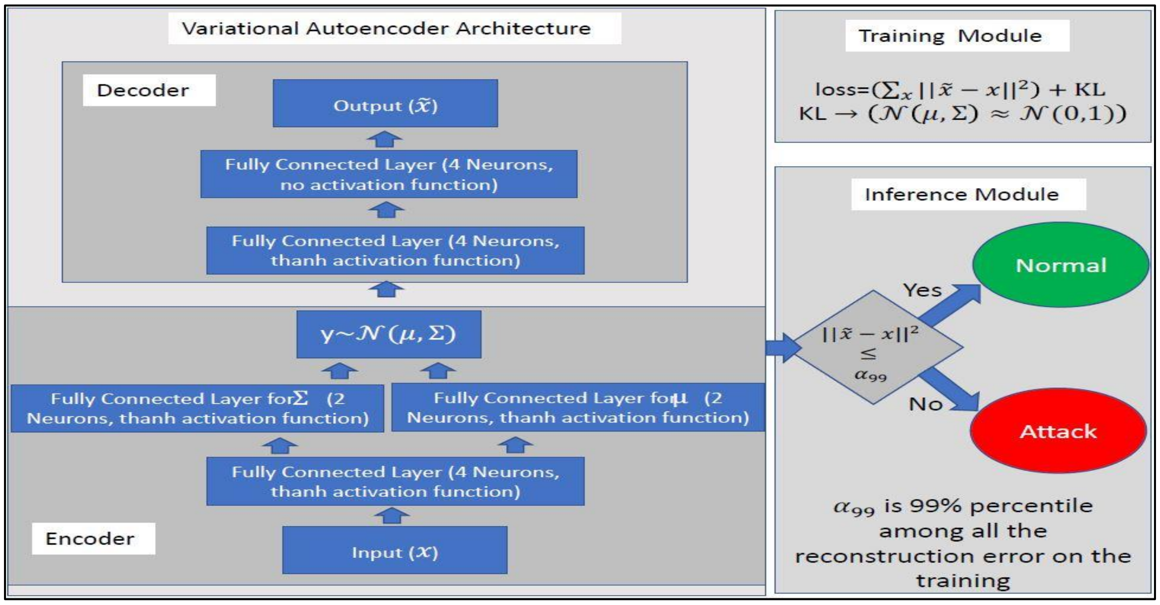

Variational Autoencoder Architecture
Decoder
Output ($\tilde{x}$)
Fully Connected Layer (4 Neurons, no activation function)
Fully Connected Layer (4 Neurons, thanh activation function)
$\gamma \sim \mathcal{N}(\mu, \Sigma)$
Fully Connected Layer for Σ (2 Neurons, thanh activation function)
Fully Connected Layer for μ (2 Neurons, thanh activation function)
Fully Connected Layer (4 Neurons, thanh activation function)
Encoder
Input (x)
Training Module
loss=$(\sum_x ||\tilde{x} - x||^2)$ + KL
KL $\rightarrow$ $(\mathcal{N}(\mu, \Sigma) \approx \mathcal{N}(0,1))$
Inference Module
$||\tilde{x} - x||^2 \leq \alpha_{99}$
Yes
No
Normal
Attack
α_{99} is 99% percentile among all the reconstruction error on the training

Figure 7 Variational Autoencoder Architecture

The Variational Autoencoder, because of regularization forces has similar input to be encoded in similar regions. This property is crucial because it allows enlarging the space of the encoded vector which becomes more suitable for attack detection. Figure 7 shows the architecture of the Variational Autoencoder and the procedure based on the reconstruction error for classifying attacks versus normal instances.

The reconstruction error by itself does not provide a complete way to classify if an example is an attack or not. Therefore, in this approach, we used the idea of the percentile threshold as described in Figure 7. By using all the reconstruction errors for each normal instance of the training set, it's computed the percentile α_{99} at 99%. The value α_{99} is used as a threshold for the reconstruction error to determine if an instance is an attack (i.e. $||x - x^{\sim}||^2 > \alpha_{99}$ or not i.e. $||x - x^{\sim}||^2 \leq \alpha_{99}$. Computing the percentile 99% as a threshold means assuming that 1% of the normal instances can be very similar to the attack instances. Note that the semi-supervised algorithms are trained only with normal instances (i.e., out anomaly dataset). Therefore, they don't require any specification in the training phase of attack instances, which makes them more suitable to detect the unknown saturation attacks.

3.3 Experiment Setup and Data Collection

3.3.1 Physical and Simulated SDN Environment

We collected the OpenFlow communication channel traffic using both physical and simulated SDN environments. The main advantage of using a physical SDN environment is the capacity to replicate the workload of a real-world SDN network and the internet traffic generated by real-world applications. Figure 8 shows the physical environment architecture, which consists of a Pica8 P-3290 OpenFlow switch, a Floodlight Master 1.2v as an SDN controller, and five hosts named from h-1 to h-5. Table 2 shows the specifications and the configurations for each host.

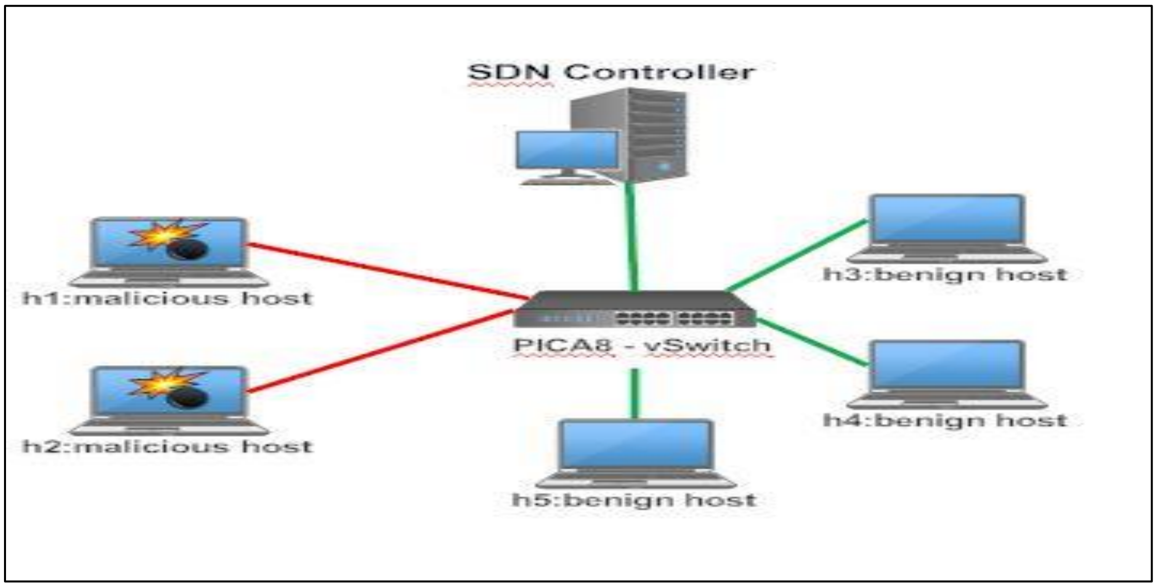


Figure 8 SDN Physical Environment Architecture

Table 2 Physical Environment Configuration and Specifications

Host Name	CPU Info	Memory Info	Operating System
Controller Machine	Intel Core (i7) 2.5GHz	16GB	Ubuntu 16.04.5 LTS
h-1	Intel Core (i5) 2.5GHz	8GB	Ubuntu 16.04.5 LTS
h-2	Intel Core (i5) 2.5GHz	8GB	Ubuntu 16.04.5 LTS
h-3	Xenon E5 2.5GHz	4GB	Ubuntu 16.04.5 LTS
h-4	Xenon E5 2.5GHz	4GB	Ubuntu 16.04.5 LTS
h-5	Intel Core (i5) 2.4GHz	8GB	Ubuntu 16.04.5 LTS

The physical environment is limited by the network scale and topology. A simulated SDN environment was created using the Mininet v2.1.0 tool [59]. The simulated environment enables the creation of different network topologies (i.e., tree topology, star topology, mesh topology, and linear topology) with a different network scale (i.e., number of hosts, number of switches). Table 3 shows the main configurations of the simulation SDN network which were obtained from the Mininet examples. Currently, Mininet runs on a single machine and simulates all the OpenFlow switches, hosts, and links in a single operating system. All of these (e.g., number of switches) share the same hardware resources, which discourage building a large-scale network and limit the capacity of the Mininet for hosting real-world applications that can mimic real network behaviors.

The proposed approach utilized the generated OpenFlow traffic of a single-controller SDN environment. However, the proposed approach could be extended to a multiple-controller SDN environment by collecting the OpenFlow traffic of each controller and generating the corresponding datasets in the same fashion.

Table 3 SDN Simulation Environment Configurations and Specifications

Parameter	**Description**	**Default Value**
Nc	Number of Controllers	1
Ns	Number of Switches	10-200
Nh	Number of Hosts	50-300
Nt	Network Topology	Star, Mesh, Ring, Tree

3.3.2 OpenFlow Traffic Generation

3.3.2.1 Benign Traffic

The benign OpenFlow traffic was collected from both physical and from simulated environments by using different traffic generation tools that mimic real-world network behaviors. For the physical environment, four tools were used to generate the OpenFlow traffic. Firstly, the D-ITG (Distributed Internet Traffic Generator) [60] was employed. D-ITG has ITGSend and ITGRecv components. ITGSend can generate parallel traffic flows and send it to different ITGRecv instances. ITGRecv is responsible for receiving traffic flows from ITGSend. D-ITG provides the ability to generate multiple unidirectional traffic flows for different protocols, such as IPv4, IPv6, TCP, UDP, ICMP, SCTP (Stream Control Transmission Protocol), DCCP (Datagram Congestion Control Protocol), DNS, Telnet, and VoIP.

Secondly, we used the Nping tool [61], which is open-source software that can generate traffic for different protocols, such as the ARP protocol. By using Nping, we were able to customize the packet size and the transmission intervals of the generated traffic.

Thirdly, in order to generate a concurrent stateful and stateless traffic that simulated the internet traffic, the Cisco's TRex realistic traffic generator [62] has been used. TRex gives us the ability to generate almost any kind of L4-7 traffic, based on the smart reply of real traffic templates. It can amplify the traffic of the server and the client-side up to 200Gb/sec.

Finally, we used OSTINATO [63] to configure and generate many traffic streams for different protocols such VLAN, IPv4, IPv6, stateless TCP, ARP, ICMPv4, ICMPv6, IGMP (Internet Group Management Protocol), MLD (Multicast Listener Discover), RTSP (Real-Time Streaming Protocol) and NNTP (Network News Transfer Protocol). For the simulated environment, we were able to use the Nping and OSTINATO tools only, due to the simulated environment limitation mentioned above.

In both environments, the Wireshark tool [64] was used to capture the OpenFlow traffic between the SDN controller and the OpenFlow switches. As shown in Table 4, the total size of the captured benign OpenFlow traffic from the physical environment was 250 GB, the total duration was about 137 hours, and the total simulated benign traffic was about 143GB, for a total duration of 100 hours.

3.3.2.2 Malicious Traffic

In the physical and simulated environments, Hping3 [65] and LOIC (Low Orbit Ion Cannon) [66] were employed to launch the saturation network attacks. The Wireshark tool was used to capture the OpenFlow malicious traffic between the SDN controller and the OpenFlow switch. Hping3 is an open-source tool for network stress testing, as well as DoS attacks. LOIC is a well-known tool used to launch DoS attacks against different agencies [67]. By using these tools, we were able to launch 31 saturation attacks that covered all combinations of SYN flooding, UDP flooding, ICMP, IP Spoofing, and SARFU-TCP flooding. In both environments, each of the attacks flooded the control and data planes. As shown in Table 4, the total size of the physical environment anomaly traffic was 50Gb and the duration for each attack was about 30 minutes. For the simulated environment, the total size of the anomaly traffic was about 100 GB and the attack duration was about 20 minutes.

Table 4 Physical and Simulated OpenFlow Traffic Description

Environment Type	Traffic Type	Number of Sessions	Duration of an individual session	Total Duration	Total Size of Captured Traffic Files
Physical Environment	Benign Traffic	104	1—4 hrs.	137 hrs.	250 Gb
	Attack Traffic	31	30 minutes	15.5 hrs.	50 Gb
Simulation Environment	Benign Traffic	100	1hr	100 hrs.	143 Gb
	Attack Traffic	31	20 minutes	10.3 hrs.	100 Gb

3.4 Evaluation Metrics

Before describing the evaluation metrics, which were used to evaluate the classifiers' detection performance for different time-windows, we should describe some terminology used in the evaluations. A True-Positive (TP) is an observation from the testing sample which has been correctly classified as an attack. A False-Positive (FP) is an observation that has been incorrectly classified as an attack, i.e., a normal observation which has been mislabeled as an attack. A False Negative (FN) is an observation that has been incorrectly classified as a normal (non-attack) observation. Finally, a True Negative (TN) is an observation that has been correctly classified as a normal observation.

In our models, we calculated the accuracy of the predictions from the K-NN, SVM, and NB models. The accuracy is defined as the total number of correct predictions divided by the total number of predictions made by the model. However, accuracy alone is typically insufficient for judging the effectiveness of a classification model [68]. Thus, instead of using the accuracy, three measures, the precision, the recall, and the F-1 score metrics have been used to evaluate the impact of different time-window traffic analysis on our model's detection performance. Precision is defined as the proportion of true-positive observations divided by the total number of true positive and false-positive observations.

Having a high precision means a low false-positive ratio, which is an important indicator of the reliability of the model's predictions. For example, a model that has 95% precision when classifying traffic samples is correct 95% of the time. The recall is defined as the ratio of the true positives to the total of true positives and false negatives. However, high recall indicates a low false-negative ratio, which is a confidence indicator of the model's ability to predict the actual positives. For instance, a model with a 90% recall can correctly identify 90% of the actual positives. Finally, the F1 score is a balance between recall and precision. Hence, for the saturation detection system, it is highly important to obtain a high precision with a high recall, and a high F1 score [68].

3.5 Experiment Results and Discussion

This research aimed to answer the following questions:

- RQ1: What is a proper time-window of OpenFlow traffic analysis for detecting saturation attacks?
- RQ2: How do different time-windows affect the detection performance of a classifier?
- RQ3: Are classifiers effective in the detection of unknown saturation attacks?

3.5.1 Proper Time-Window for Detection of Known Attacks

This section describes the experiments that were conducted to discover the proper time-window for OpenFlow traffic analysis to detect the saturation network attacks. Also, the impact of different time-windows of OpenFlow traffic analysis on the detection performance of the known saturation network attacks for SVM, K-NN classifiers. We refer here to the known saturation attack as an attack that has been included in the training phase, in other words, the training dataset has many samples that describe the attack behavior and our classifier models have been trained to detect this attack.

In the physical environment, 30 datasets were generated from the collected OpenFlow traffic and each dataset represents a different time-window, ranging from 1 minute to 30 minutes. In our experiments, each dataset is used to train and test the K-NN, the SVM, the NB models and we used the precision, recall, and F1 score metrics to evaluate the performance of our models and analyze the impact of different time-windows on the model's prediction results. We now discuss how we determined the earliest proper time-window for each classifier, from these experimental results.

Figure 9 shows the precision, recall, and F1 score metrics for the K-NN models when the time-window ranges from 1 minute to 30 minutes. The highest detection rate for the K-NN classifier is obtained when the time-window is equal to 1 minute: the precision is 96%, with recall 95%, and the F1 score is 95%. The lowest detection rate is when the time-window is equal to 30-minutes in which case, the corresponding precision is 47%, with a recall of 98%, and the F1 score is 64%. As a result, in the physical environment, the optimal time-window for the K-NN classifier is one minute of traffic analysis. To support this conclusion, Figure 9 shows that the precision and F-1 score decrease as the time-window of traffic analysis increases.

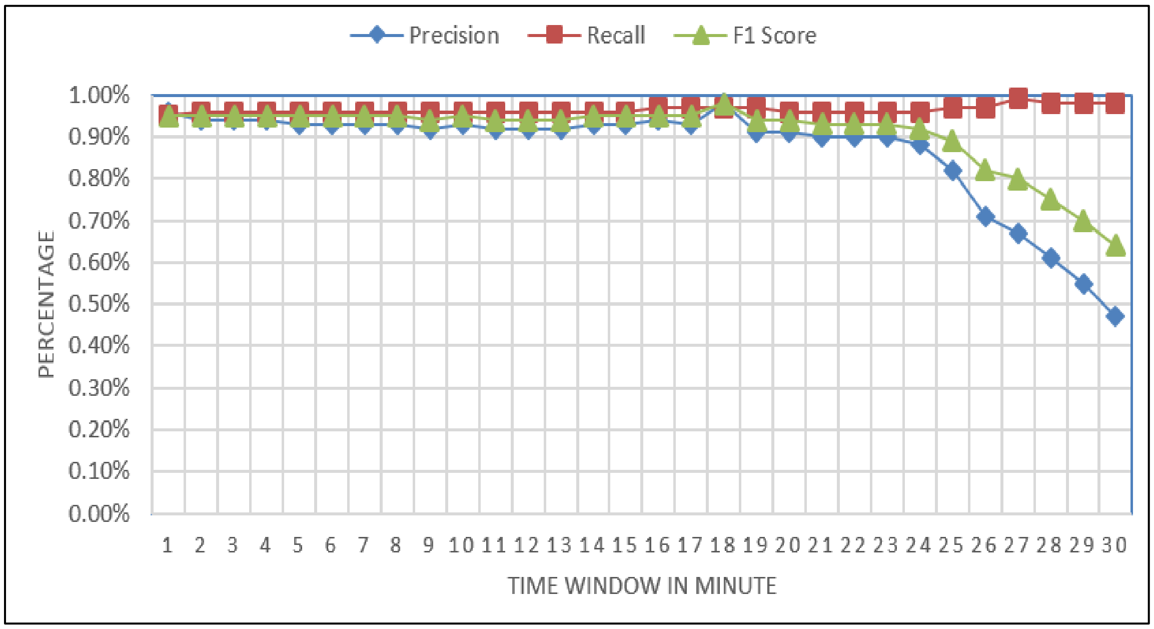


Figure 9 K-NN Precision, Recall, and F1 Score with Different Time Windows Using Physical Environment

Figure 10 shows the values of the evaluation metrics for the SVM models. The highest detection result is when the time-window equals 1 minute; in this case, the precision is 91%, with a recall of 91%, and the F1 score is 91%. In contrast, the lowest precision achieved by these models is 46%, with a recall of 99%, and an F1 score of 62%. This is when the time-window is 30 minutes. Figure 10 shows the impact of time-window length on the SVM classifier performance. Notably, the precision and F1 score decrease and the recall increase as the time-window increases. Thus, the optimal time-window for the SVM classifier in the physical environment is also 1 minute.

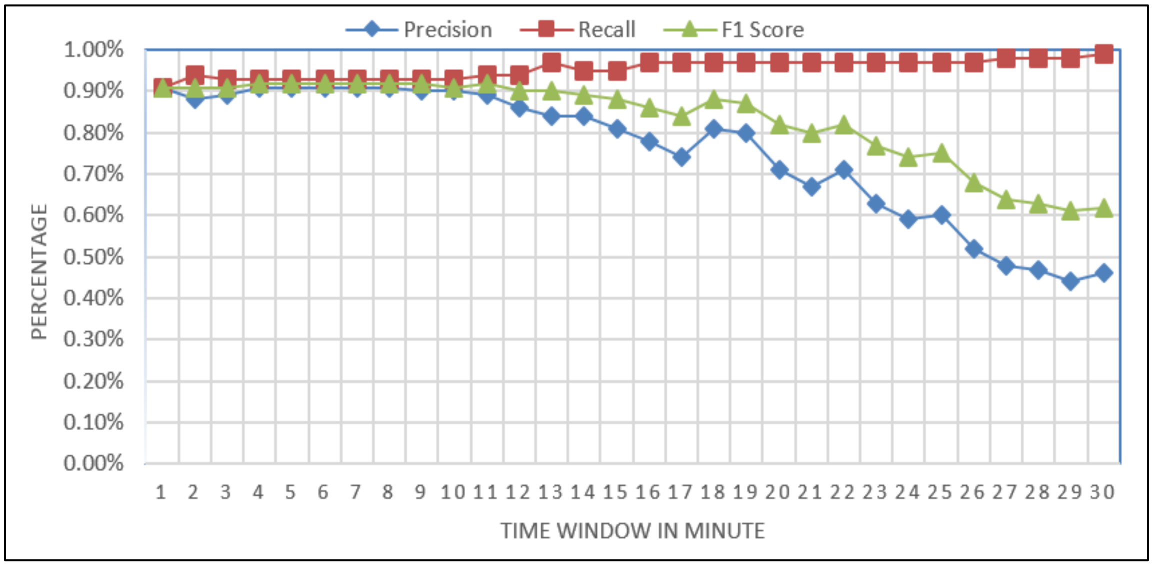


Figure 10 SVM Precision, Recall, and F1 Score on with Time Windows Using Physical Environment

Similar to the approach for the K-NN and SVM classifiers, we performed 30 experiments to evaluate the NB classifier detection performance for the same time-windows. Figure 11 shows the results of our evaluation. The highest precision is 99%, with a recall of 80%, and an F1 score of 89%, when the time-window equals three minutes. The lowest precision is 52%, with a recall of 53%, and an F1 score of 52% when the time window equals 30 minutes. In the physical environment, the 3-minute time-window seems optimal for the NB classifier in order to obtain a high detection rate.

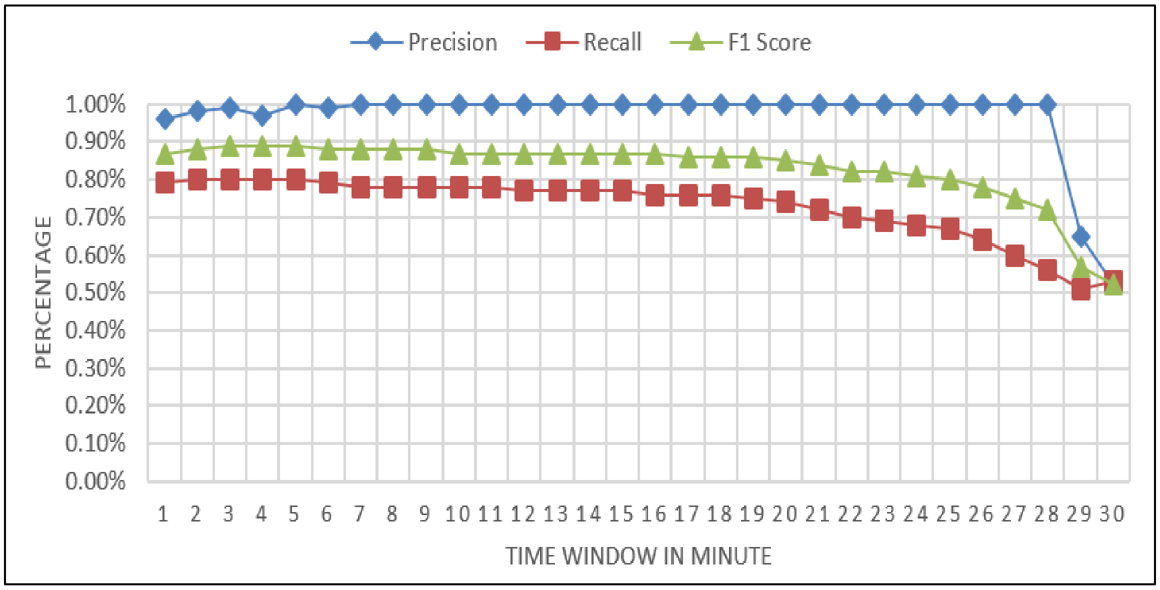


Figure 11 NB Precision, Recall, and F1 Score with Different Time Windows Using Physical Environment

In the simulation environment, 20 datasets were generated and used in training and testing the proposed classifiers. Similar to the physical environment experiments, each dataset represented a different time-window of traffic analysis. In this case, the dataset time-windows ranged from 1 minute to 20 minutes. Again, we used the precision, recall, and F1 score to evaluate the performance of our classifiers in each experiment.

Figure 12 shows that when the time-window equals 1 minute, the K-NN classifier achieves a high precision of 97%, with a recall of 99%, and an F-1 score of 98%. When the time-window equals 18 minutes, the K-NN classifier achieves the highest precision overall (100%) but suffers from low recall (19%) and a low F1 score (35%). As a result, we conclude that the 1-minute time-window is optimal for the K-NN classifier, in order to obtain the highest detection results in the simulation environment. Figure 12 shows the impact of the time window on the detection performance of the K-NN classifier. Increasing the time-window led to increased precision but decreased the recall and the F1 score ratios.

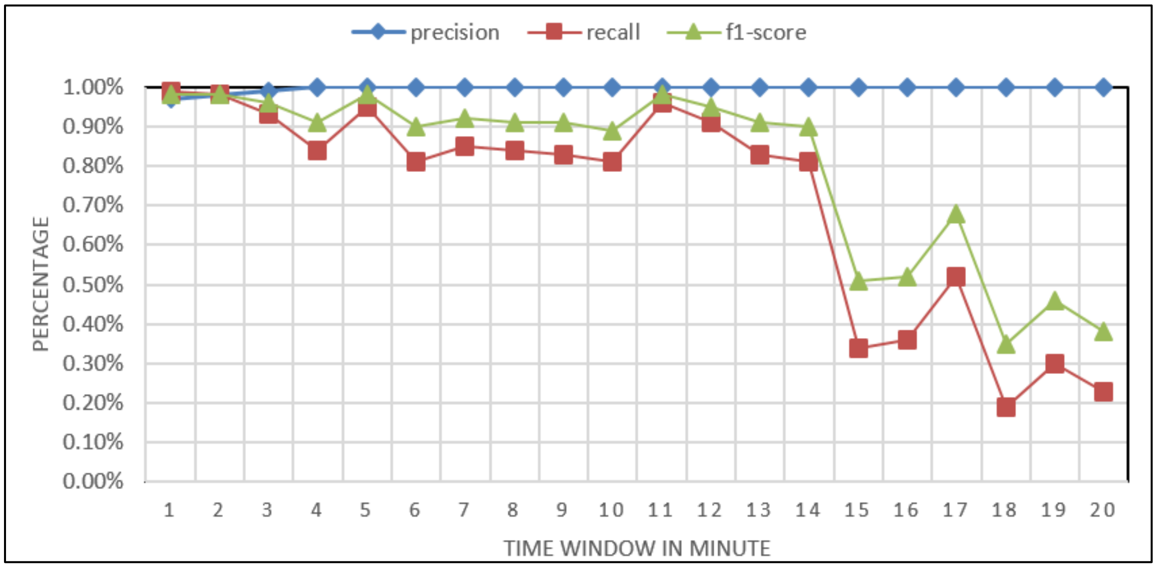


Figure 12 K-NN Evaluation Metrics Result with Different Time Windows Using Simulation Environment

As shown in Figure 13, the SVM classifier achieved the highest detection results with a time-window of 2 minutes. The corresponding precision is 81%, with a recall of 89%, and the F1 score is 85%. Moreover, Figure 13 shows noticeable changes in the SVM classifiers' detection performance when the time-window increases. For example, when the time-window equals 20 minutes, the detection results declined significantly to a precision of 7%, recall of 35%, and an F1 score of 11%.

Figure 14 shows the metric values for the NB classifier model results. The NB model obtained the highest detection results when the time-window was 1 minute, with corresponding precision of 85%, recall of 96%, F1 score of 91%. The lowest precision was 11%, with a 37% recall rate, and an F1 score of 17%. In our simulation environments, we found that the 1-minute time-window is optimal for the NB classifier to detect the attack traffic. The experimental results show the critical role that the time-window of OpenFlow traffic analysis plays on the detection performance of our machine learning classifiers. In the upcoming section, we describe our investigation and share our findings on the impact of the time window on the detection performance of our classifiers.

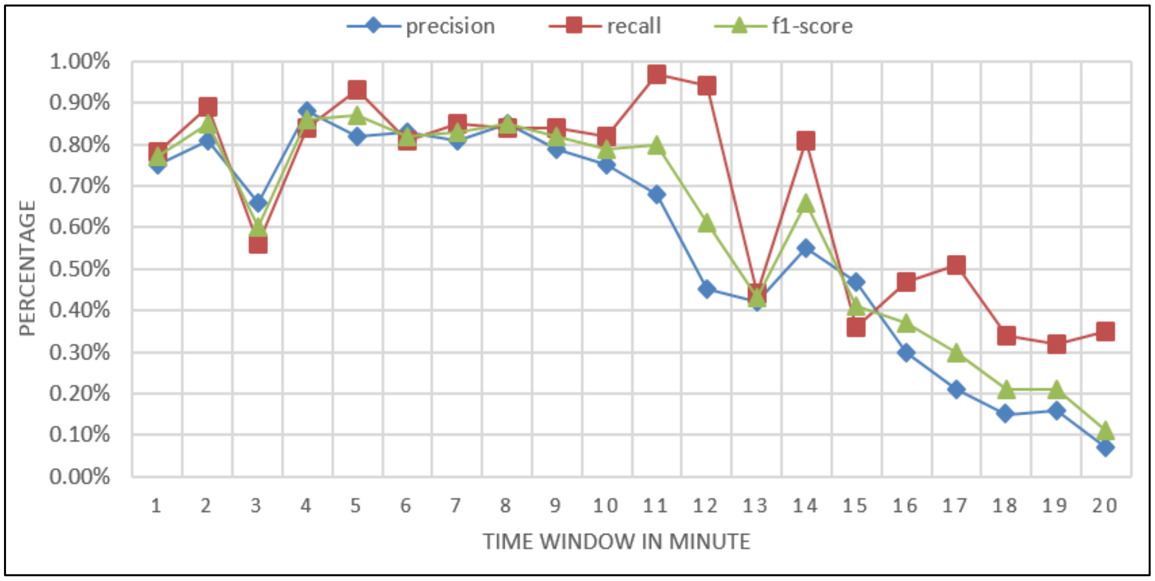


Figure 13 SVM Evaluation Metrics Result with Different Time Windows Using Simulation Environment

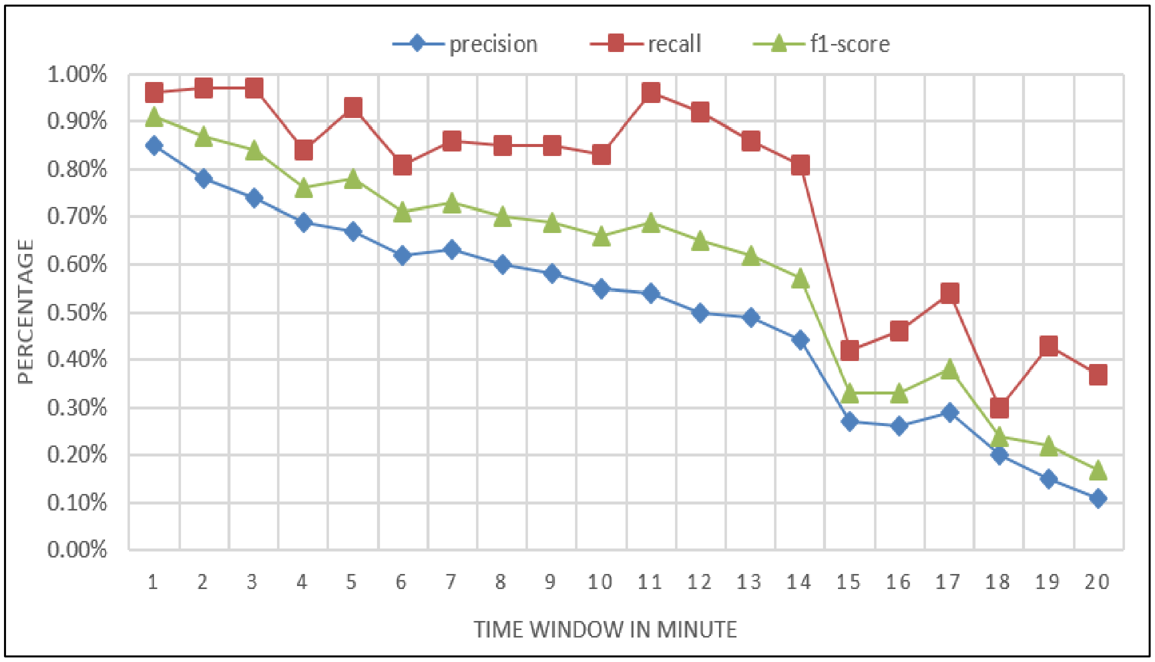


Figure 14 NB Evaluation Metrics Result with Different Time Windows Using Simulation Environment

3.5.2 Impact of Time-Window Variations

To better quantify the impact of different time-windows of OpenFlow traffic analysis on the detection performance of the K-NN, SVM, and NB classifiers, we conducted a false-negative analysis. The purpose of a false-negative analysis is to find all the samples that are falsely identified by our classifiers and the time slot of each sample. In this application, the 'false negative' samples represent the attack samples that were falsely identified as benign samples by the trained models.

We conducted the false-negative analysis, after performing the physical and simulated environment experiments, by allocating the attack samples for each experiment dataset with corresponding trained K-NN, SVM, and NB models, wherein, each experiment dataset represents a different time-window of OpenFlow traffic analysis. Therefore, the attacking samples were fed to the trained models and extracted the samples that were falsely identified by the models as a normal sample along with the time slot of that sample**.** Based on the false-negative analysis results, we discovered that most of the false negatives occurred at the end of the attack time.

As shown in Figures 9-14, the recall ratios, precision ratios, and F1 scores in the experiments on both environments decreased significantly when the time-window was equal to 15 minutes or longer. The reason behind the increasing number of false negatives when the time-window increased is due to the behavior of the SDN environment when it is under a saturation attack. Technically, in the early stages of a saturation attack (i.e. when the attack is initiated), both the switch and the controller have enough capacity to process the incoming attack packets. Also, the OpenFlow connection channel has sufficient bandwidth to transfer the OpenFlow messages at this time. This leads to a significant increase in the numbers of Packet-In, Packet-Out, Packet-Mod, and TCP-ACK messages in the OpenFlow traffic. In this situation, the K-NN, the SVM, and the NB classifiers were able to accurately identify the attack samples from the benign samples, as evidenced by the high precision ratios, recall ratios, and F1 scores of the classifiers.

Subsequently, as the attack takes over the SDN network, the OpenFlow switch and the controller become overwhelmed. At this point, they do not have sufficient capacity to process the huge amount of malicious traffic, and the OpenFlow channel is also congested. Thus, the occurrences of Packet-In, Packet-Out, Packet-Mod, and TCP-ACK messages in the malicious OpenFlow traffic are similar to the occurrences of these messages in the benign OpenFlow traffic. Therefore, the K-NN, SVM, and NB classifiers are more likely to falsely identify the attack samples that are similar to benign samples as normal samples, which leads to an increase in the number of false negatives. This, in turn, reduces the recall ratios. They may also falsely identify the benign samples as attack samples, which in turn increases the false positives and decreases the precision ratio. As a result, the overall detection performance of the machine learning classifier suffers.

3.5.3 Detection of Unknown Attacks

An unknown attack is an attack that has been mislabeled by the training model due to the absence of similar samples in the training dataset. In our experiments, we excluded the targeted attack and its combination of samples from the training dataset, in order to present it as an unknown attack to the trained model. In this case, the training dataset included benign traffic samples, as well as the remaining attacks and their respective samples.

The testing dataset consisted of the unknown attack samples, as well as randomly selected benign traffic samples. For example, given that X is the training dataset that consists of attack samples and normal traffic samples, Y is the testing dataset that includes attack samples and normal traffic samples, G is the unknown attack samples only, and C is the attack combination samples, the X training datasets and Y testing datasets are in the form of:

$$X_{(\text{trainingSet})} = X - G - C \tag{3.1}$$

$$Y_{(\text{testingSet})} = G + NormalTrafficSamples \tag{3.2}$$

In addition, we studied the impact of different time-windows of traffic analysis on the detection results by selecting the proper time-window of OpenFlow traffic analysis for each classifier in each environment, based on the previous experimental results. Figure 15 summarizes the detection performance results of our classifiers in both physical and simulated SDN environments.

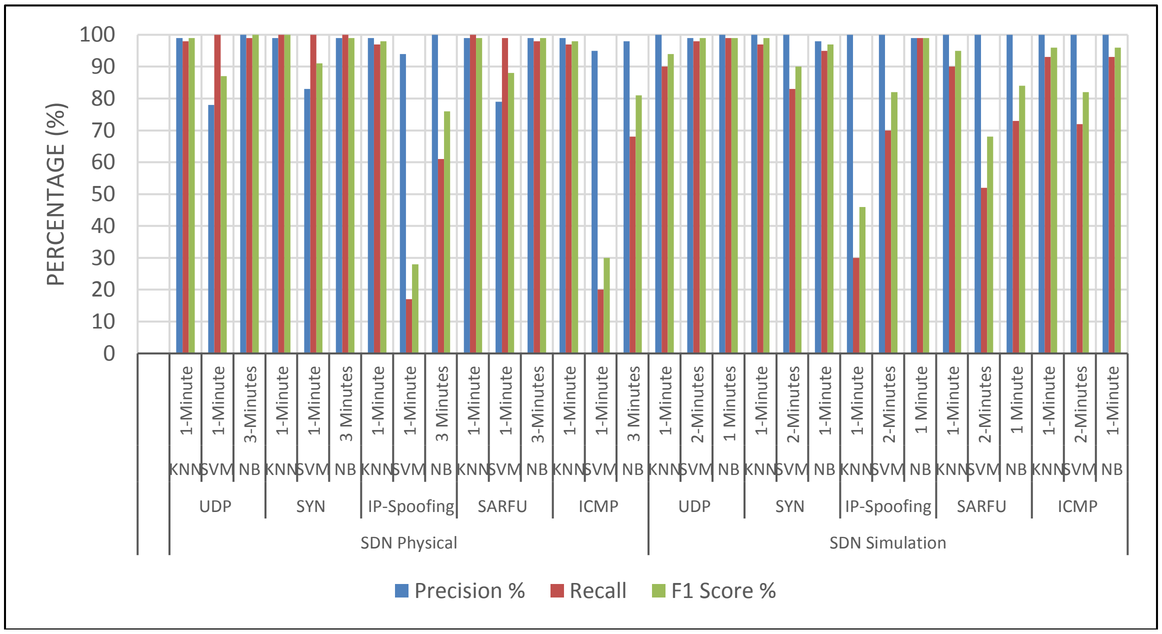


Figure 15 Unknown Saturation Attacks Detection Results

Based on the reported results, the classifiers are capable of detecting the unknown saturation attacks in both environments. In particular, the K-NN classifier shows promising detection results in both SDN environments. Also, the reported results show that the detection performance of the classifiers was influenced by the SDN environment setup. For instance, the SVM classifier obtained a 78% precision ratio, a 17% recall ratio, and a 28% F-1 score for detecting the IP-Spoofing attacks in the physical environment, whereas, it obtained 100% precision, 70% recall, and an 82% F-1 score in the simulated environment for the same attack.

Based on these results, we believe that there is a relationship between the SDN environment setup and the detection performance of our classifiers. For instance, the SVM classifier obtained a 78% precision ratio, a 17% recall ratio, and a 28% F-1 score for detecting the IP-Spoofing attacks in the physical environment, whereas, it obtained 10% precision, 70% recall, and an 82% F-1 score in the simulated environment for the same attack. We hypothesize that all the different attacks in the training set allow the classifier to generalize the one missing. To confirm this hypothesis, we consider the K-NN (our best choice) in the case where only one kind of attack is used in the training set.

The results in Table 5 show that the K-NN in the worst-case scenario obtained a low detection performance result in detecting the unknown saturation attacks. This means that supervised classification cannot be trusted for unknown SDN attacks in general. For this reason, we consider semi-supervised classifiers such as Isolation Forest, One Class-SVM, Basic Autoencoder, and Variational Autoencoders. In this case, the training set comprises only the normal instances, whereas, the testing set consists of all the normal and attack instances. Table 6 demonstrates the average results of the 10-fold cross-validation of the semi-supervised classifiers.

The semi-supervised algorithms have higher detection performance results of unknown saturation attacks than the supervised ones. Specifically, the Variational Autoencoder is effective in detecting unknown saturation attacks and obtains comparable results to the supervised classifiers in detecting the known saturation attacks. Thus, in this approach, the online saturation attack detection module utilized the Variational Autoencoder algorithm as a machine learning classifier to detect the known and unknown saturation attacks in SDN.

Table 5 K-NN Unknown Detection Result of One Attack Training

Attack	**Physical Environment**			**Simulated Environment**		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
UDP	0.99	0.23	0.38	0.95	0.80	0.86
SYN	0.96	0.21	0.41	0.96	0.77	0.85
SARFU	0.99	0.24	0.38	0.96	0.77	0.85
ICMP	1.00	0.30	0.30	0.95	0.82	0.87
IP-Spoofing	0.97	0.13	0.22	0.96	0.74	0.83

Table 6 Semi-Supervised Detection Results

Algorithm	**Physical Environment**			**Simulated Environment**		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Isolation Forest	0.56	0.92	0.69	0.38	1.00	0.56
One-Class SVM	0.22	0.97	0.35	0.116	1.00	0.20
Basic Autoencoder	0.99	0.24	0.38	0.96	0.80	0.86
Variational Autoencoder	0.86	0.93	0.90	0.85	0.97	0.91

Nonetheless, we have demonstrated that the classifiers are capable of detecting unknown saturation attacks with reasonable accuracy. We believe this is due to several characteristics of the problem. For (1), the saturation network attacks have a high degree of self-similarity. [69] studied the self-similarity characteristics of benign and malicious OpenFlow traffic. Their results show that the normal OpenFlow traffic has a low degree of self-similarity and has different statistical characteristics, whereas, the saturation attacks on OpenFlow traffic have a higher degree of self-similarity. (2) Our features can accurately reflect abnormal behavior within OpenFlow traffic because they represent the main messages of the OpenFlow v1.5 protocol. In other words, the models are sensitive to any abnormal activity that occurs in the OpenFlow traffic between the control and data planes, because this activity is encoded in the features we have chosen for our datasets. Essentially, all the saturation attacks exhibit a similar technique of flooding the SDN environment by generating a vast number of table-miss packets; therefore, they have a similar impact on the OpenFlow messages.

3.6 Summary

In this chapter, we have studied the K-NN, SVM, and NB classifiers for the detection of saturation attacks in physical and simulated SDN environments. The experiment results have demonstrated that the time window of OpenFlow traffic has a noticeable impact on the detection performance and that the classifiers were capable of detecting known types of saturation attacks in SDN.

Also, we have investigated the capability of semi-supervised and supervised classifiers for detecting unknown saturation attacks. Based on the reported results, the supervised classifiers such as K-NN, SVM, and NB are deficient in detecting unknown saturation attacks, whereas the semi-supervised classifiers such as One-Class SVM, Isolation Forest, Basic Autoencoder, and Variational Autoencoder can detect both known and unknown saturation attacks. Specifically, the Variational Autoencoder obtained high performance in detecting saturation attacks against SDNs.

CHAPTER FOUR: VICTIM SWITCH DETECTION

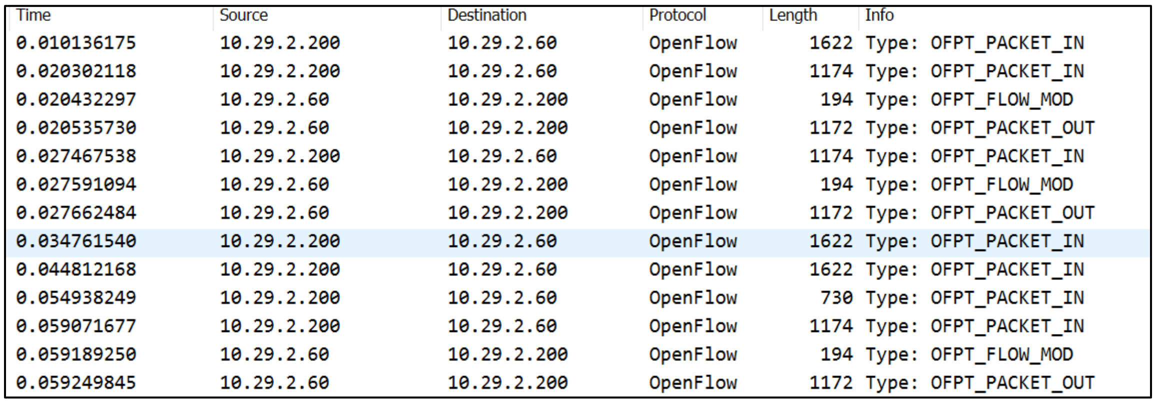
The victim switch detection module is triggered as soon as the saturation attack detection module (see Chapter 3) determines that the SDN network is under a saturation attack. It is responsible for identifying the OpenFlow switches targeted by known and unknown saturation attacks. Determining which OpenFlow switch in an SDN network is targeted by a saturation attack is an issue. In a traditional network, the task of identifying the victim switch is simpler. For example, in the case of a traditional network, determining whether a switch is compromised or not only requires examining its forwarding behaviors. If the forwarding behaviors diverge from the switch's predefined forwarding rules, then the switch is compromised. However, in an SDN, different controller applications and modules are involved in the programming of the OpenFlow switches. Thus, the flow-rules of an OpenFlow switch dynamically change over time. Consequently, the forwarding behaviors of an OpenFlow switch do not exhibit a single set of behavioral norms, which makes the task of identifying the victim switch difficult.

Furthermore, a table-miss occurs because of a new incoming benign packet or a malicious packet. Neither one matches any of the OpenFlow switch flow-entries. In both cases, the OpenFlow switch behavior is identical: it generates a Packet-In message and forwards it to the controller. Next, the controller decides on the proper action to deal with the table-miss packet, either installing a new flow-entry into the switch flow-table using a Packet-Mod message and allocating the table-miss packet route using a Packet-Out message. These steps occur without the controller knowing if the table-miss packet is a malicious one or not.

In short, the behavior of the OpenFlow switch and the controller when a table-miss occurs is the same, whether the table-miss is due to a malicious packet or due to a new benign packet. This makes the determination of the victim OpenFlow switch a complex issue.

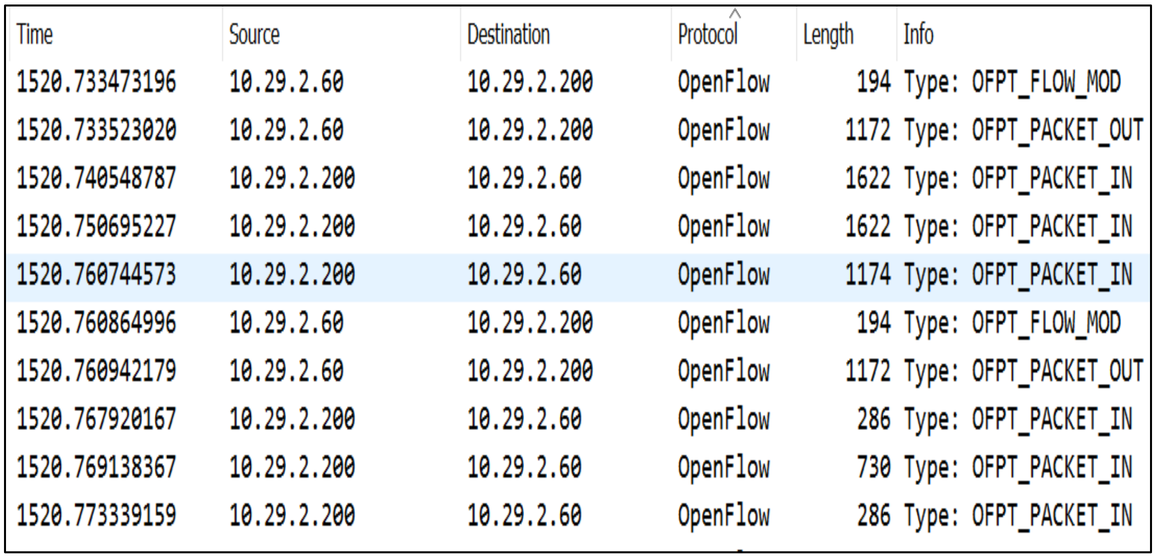
To complicate the matter, malicious OpenFlow traffic generated by a saturation attack, and benign OpenFlow traffic generated using a normal traffic generation tool or any SDN application, contain the same OpenFlow messages.

Figures 16 and 17 show samples of benign and malicious OpenFlow traffic that consist of the OpenFlow messages that have been transferred between the controller and the OpenFlow switches. The benign OpenFlow traffic was generated using the D-ITG and Cisco Trex normal traffic generation tools and the malicious OpenFlow traffic was generated using the HPING3 flooding tool. From these two samples, we can see the similarity in the OpenFlow messages between the malicious and the benign. These samples also highlight the lack of unique features that could be used to distinguish malicious OpenFlow traffic from legitimate traffic.



Time	Source	Destination	Protocol	Length	Info
0.010136175	10.29.2.200	10.29.2.60	OpenFlow	1622	Type: OFPT_PACKET_IN
0.020302118	10.29.2.200	10.29.2.60	OpenFlow	1174	Type: OFPT_PACKET_IN
0.020432297	10.29.2.60	10.29.2.200	OpenFlow	194	Type: OFPT_FLOW_MOD
0.020535730	10.29.2.60	10.29.2.200	OpenFlow	1172	Type: OFPT_PACKET_OUT
0.027467538	10.29.2.200	10.29.2.60	OpenFlow	1174	Type: OFPT_PACKET_IN
0.027591094	10.29.2.60	10.29.2.200	OpenFlow	194	Type: OFPT_FLOW_MOD
0.027662484	10.29.2.60	10.29.2.200	OpenFlow	1172	Type: OFPT_PACKET_OUT
0.034761540	10.29.2.200	10.29.2.60	OpenFlow	1622	Type: OFPT_PACKET_IN
0.044812168	10.29.2.200	10.29.2.60	OpenFlow	1622	Type: OFPT_PACKET_IN
0.054938249	10.29.2.200	10.29.2.60	OpenFlow	730	Type: OFPT_PACKET_IN
0.059071677	10.29.2.200	10.29.2.60	OpenFlow	1174	Type: OFPT_PACKET_IN
0.059189250	10.29.2.60	10.29.2.200	OpenFlow	194	Type: OFPT_FLOW_MOD
0.059249845	10.29.2.60	10.29.2.200	OpenFlow	1172	Type: OFPT_PACKET_OUT

Figure 16 OpenFlow Benign Traffic



Time	Source	Destination	Protocol	Length	Info
1520.733473196	10.29.2.60	10.29.2.200	OpenFlow	194	Type: OFPT_FLOW_MOD
1520.733523020	10.29.2.60	10.29.2.200	OpenFlow	1172	Type: OFPT_PACKET_OUT
1520.740548787	10.29.2.200	10.29.2.60	OpenFlow	1622	Type: OFPT_PACKET_IN
1520.750695227	10.29.2.200	10.29.2.60	OpenFlow	1622	Type: OFPT_PACKET_IN
1520.760744573	10.29.2.200	10.29.2.60	OpenFlow	1174	Type: OFPT_PACKET_IN
1520.760864996	10.29.2.60	10.29.2.200	OpenFlow	194	Type: OFPT_FLOW_MOD
1520.760942179	10.29.2.60	10.29.2.200	OpenFlow	1172	Type: OFPT_PACKET_OUT
1520.767920167	10.29.2.200	10.29.2.60	OpenFlow	286	Type: OFPT_PACKET_IN
1520.769138367	10.29.2.200	10.29.2.60	OpenFlow	730	Type: OFPT_PACKET_IN
1520.773339159	10.29.2.200	10.29.2.60	OpenFlow	286	Type: OFPT_PACKET_IN

Figure 17 OpenFlow Malicious Traffic

In this chapter, we study the adoption of supervised and semi-supervised machine learning algorithms with three methods of victim switch detection to determine the most accurate and effective approach for identifying the victim OpenFlow switches, as explained below.

4.1 Victim Switch Detection Using OpenFlow Messages Header

This is a lightweight method that can be used to detect the victim OpenFlow switches by inspecting OpenFlow message headers. This method works as follows. Firstly, classify the OpenFlow messages based on the OpenFlow switch DPID. Secondly, by inspecting the OpenFlow message headers, extract each message type. Finally, based on message type, extract the following features: (1) number of Packet-In messages generated by a switch, (2) number of Packet-Out messages received by a switch, (3) number of Packet-Mod messages received by a switch, and (4) number of TCP-ACK messages received and generated by an OpenFlow switch.

Essentially, the attackers may compromise an OpenFlow switch by sending a vast number of table-miss packets. Thus, a large number of Packet-In messages will be generated by the targeted switch and forwarded to the controller. Therefore, the occurrences of Packet-In messages in the malicious OpenFlow traffic of the targeted OpenFlow switch are much more frequent, because many incoming packets do not match the existing flow-entries of the targeted OpenFlow switch.

In addition, in the benign traffic, the number of Packet-Out and Packet-Mod messages is very similar to the number of Packet-In messages and the number of TCP-ACK messages. In contrast, in the malicious traffic, the number of Packet-In messages and TCP-ACK messages is significantly higher than the number of Packet-Out and Packet-Mod messages, as depicted in Figure 18. Thus, these features can be used to identify the victim OpenFlow switches because they reflect the impact of the saturation attack on the OpenFlow traffic of the targeted OpenFlow switches.

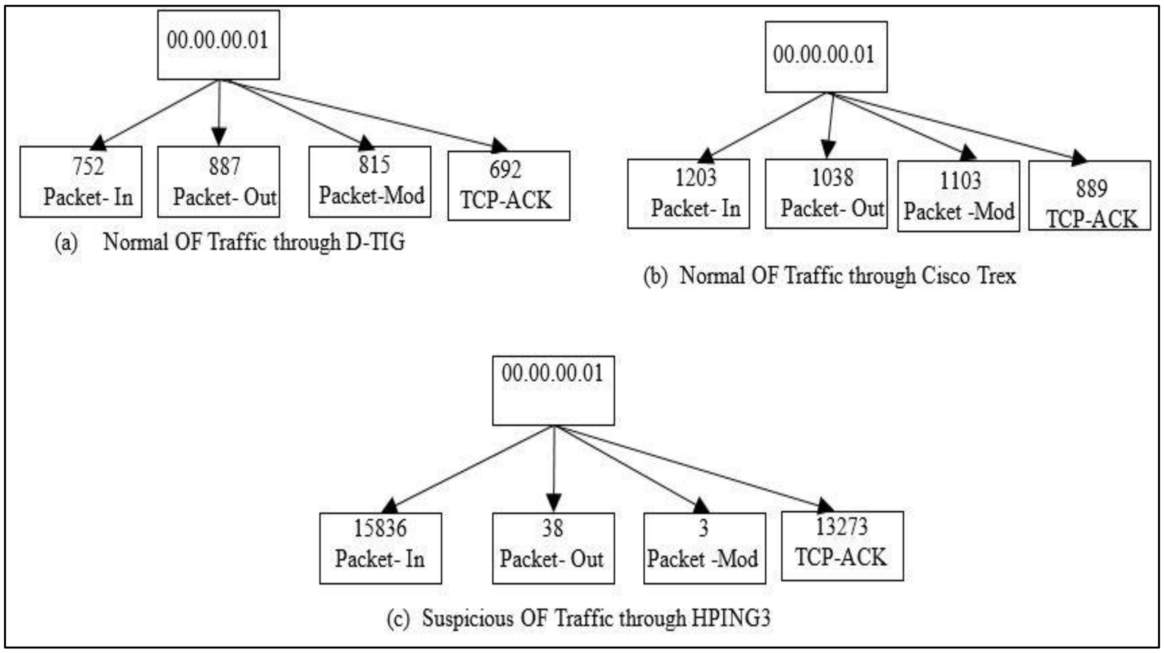


Figure 18 A Sample of the Number of Packet-In, Packet-Out, Packet-Mod, and TCP-ACK Messages in Benign and Malicious Traffic

4.1.1 Data Collection and Experiment Setup

The simulated OpenFlow traffic has been used to extract the training and testing datasets since they were collected from different network topologies and scales. Also, the malicious OpenFlow traffic was collected by targeting multiple OpenFlow switches by saturation and hybrid saturation attacks. Thus, the simulated SDN OpenFlow traffic reflects the real-world malicious OpenFlow traffic.

As mentioned in the previous section, the features of the extracted datasets are: (1) number of Packet-In messages for each OpenFlow switch, (2) number of Packet-Out messages for each OpenFlow switch, (3) number of Packet-Mod messages for each OpenFlow switch, and (4) number of TCP-ACK messages for each OpenFlow switch. The time window of the extracted datasets is equal to 1 minute of OpenFlow traffic analysis, based on the findings in Chapter 3.

We evaluated the effectiveness of this method for identifying the OpenFlow switches targeted by known saturation attacks by incorporating the K-NN, SVM, and NB classifiers. Also, we evaluated the degree of consistency of this method with the saturation attack detection method by performing 31 tests that include all the saturation and hybrid saturation attacks.

Each test consists of two parts. First, we take a random sample of the saturation attack detection method testing dataset. This sample could be a normal sample that reflects that the SDN environment is in its normal mode or an attack sample that represents that the SDN environment is under a saturation attack. Second, we take a random sample of the victim switch detection method testing dataset, which is used to evaluate the detection performance of the classifiers in distinguishing the OpenFlow switches targeted by saturation attacks from the normal ones. This sample consists of a list of normal and malicious OpenFlow switch specimens.

4.1.2 Experiment Results

Table 7 shows our offline supervised classifiers' experimental results. The K-NN classifier obtained the highest detection performance for detecting the victim OpenFlow switches, with 91% precision, 89% recall, and a 90% F1 score, which is a relatively low detection performance result. Also, the results of our victim switch models were inconsistent with the detection module models in seven tests out of 31 tests. The reason is related to the extracted features of this method, which provide too shallow of a distinction to the trained models to distinguish the targeted OpenFlow switches from the normal ones.

Table 7 Detection Results of Known Saturation Attacks-Using OpenFlow Message Header

Algorithm	Precision	Recall	F1-Score
K-NN	91%	89%	90%
SVM	82%	77%	79%
NB	86%	80%	83%

Essentially, on a large scale SDN network, many network applications generate a burst of new traffic that does not match the existing flow-rules of OpenFlow switches which cause a table-miss for a period of time. Thus, the occurrences of the Packet-In, Packet-Out, Packet-Mod, and TCP-ACK messages of normal traffic will be similar to the malicious OpenFlow traffic of the corresponding OpenFlow switch.

Also, the behavior of an OpenFlow switch when a table-miss occurred due to a normal or a malicious packet is identical because of the generating Packet-In message injecting the table-miss inside the Packet-In data field. Therefore, the headers of normal and malicious OpenFlow messages processed by the OpenFlow switches and the controller are identical. Thus, using the headers of the OpenFlow messages is not a preferable method to accurately identify the targeted OpenFlow switches. Therefore, the adoption of other methods was investigated as explained below.

4.2 Victim Switch Detection Using OpenFlow Messages Payload

Based on the investigation of the malicious and normal OpenFlow traffic, we have observed that when an OpenFlow switch is under a saturation attack, the distribution of the source IPv4 addresses of the table-miss packets that were encapsulated inside the payload of the Packet-In message (i.e., data field) change frequently. Therefore, a statistical method should be utilized to measure the distribution changes of the source IPv4 addresses of the OpenFlow switch table-miss packets.

We obtained the Packet-In messages for each OpenFlow switch and then inspected each Packet-In message payload in order to extract the table-miss packet source IPv4 addresses. Subsequently, we used the Shannon Entropy [70] to calculate the entropy value of the source IPv4 addresses of the table-miss packets for each OpenFlow switch. The Shannon Entropy is a measure of the uncertainty of random variables in information theory.

A high entropy value indicates a more decentralized probability distribution, while a low entropy value indicates a more concentrated distribution. According to the definition of the Shannon Entropy, the entropy value of the source IPv4 addresses of the switch table-miss packets can be defined as:

$$E(srcIP) = -\sum_{i=1}^{k} (n_i/M)\log_2(n_i/M) \tag{4.1}$$

Here, $srcIP = \{n_1, n_2, \ldots, n_k\}$ represents all the source IPv4 addresses of the switch table-miss packets encapsulated inside the switch Packet-In messages within the specified time window. n_i represents the occurrence number of the ith source IPv4 address, IP_i, and k is the number of different sources IPv4 addresses. $M = \sum_{i=1}^{k} n_i$ is the total occurrence number of all source IPv4 addresses of table-miss packets of a victim OpenFlow switch.

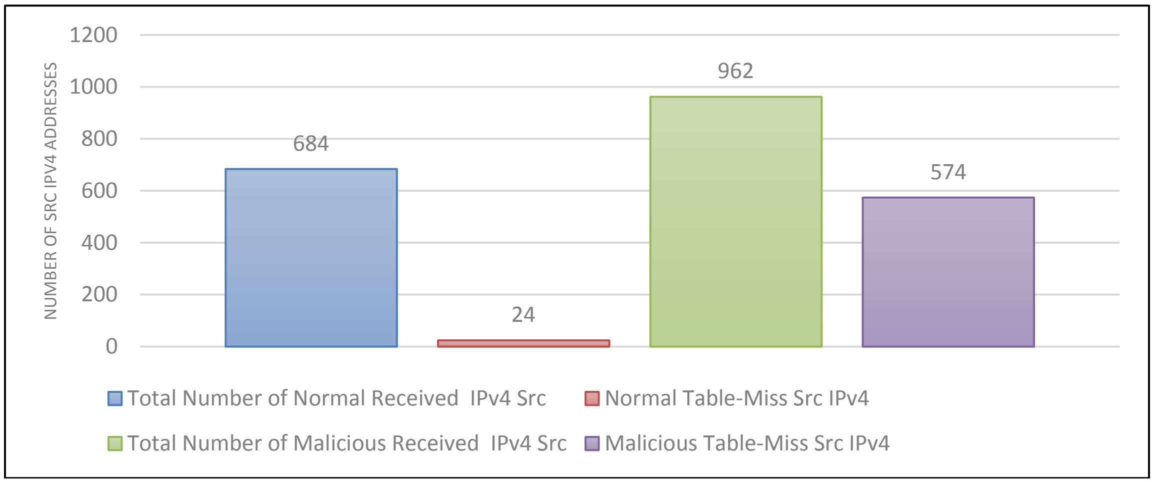


Figure 19 A Sample of Source IPv4 Addresses of Normal and Malicious Table-Miss Packets for One Minute

Figure 19 shows a sample of the total received source IPv4 addresses of normal and malicious table-miss packets. Also, it shows the number of source IPv4 addresses corresponding to the normal and malicious table-miss packets. Within one minute of OpenFlow traffic analysis, out of 962 source IPv4 addresses corresponding to the malicious incoming packets, 574 different spoofed source IPv4 addresses were extracted and used to generate table-miss packets. In contrast, out of 684 IPv4 source addresses corresponding to the legitimate incoming packets, 24 different source IPv4 addresses were extracted. Thus, the entropy value of the malicious table-miss packets was higher than the entropy value of the legitimate table-miss packets. Therefore, we have used the entropy value of the source IPv4 addresses of the OpenFlow switch table-miss packets as a feature for the machine learning classifiers to identify the targeted OpenFlow switch. The entropy value is useful for detecting the victim OpenFlow switches because it accurately reflects the characteristics of the saturation attacks against the SDN-data plane.

We have also identified another feature that can be used to accurately identify the victim OpenFlow switches. Table-Miss Packet Rate (TPR) is a new feature that has been identified in this research. It can be used to calculate the Table-Miss Packet Rate Value (TPR-Value) which is the proportion of table-miss packets (i.e., Packet-In messages) out of the total corresponding received packets of an OpenFlow switch within a specified time window. It is defined as follows:

$$TPR - Value = \frac{\sum S(PacketIn)}{\sum S(Received\ Packets)} \tag{4.2}$$

Here, $\sum S(PacketIn)$ is the total number of generated Packet-In messages, which is equal to the number of table-miss packets, and $\sum S(Received\ Packets)$ is the total number of received/incoming packets of the OpenFlow switch within the specified time window.

The TPR-value is a significant feature that can be used to accurately identify the victim switch since it measures the ratio of the received packets that cause the table-miss of an OpenFlow switch within the specified time window. As shown in Figure 20, within one minute of OpenFlow traffic analysis, the switch received 140,531 legitimate packets and generated 15,637 Packet-In messages, with a TPR-value equal to 0.11 (15,637/140,531). Therefore, the TPR-value indicates that, within one minute, 11% of the received packets caused a table-miss because they did not match any of the OpenFlow switch flow-entries, and 89% of the received packets matched the flow-entries. In contrast, the same OpenFlow switch targeted by saturation attacks received 96,463 packets and generated 38,164 Packet-In messages, with a TPR-value equal to 0.39 (38,164/96,463).

This means that 39% of the received packets are table-miss packets and 61% of the malicious packets match the flow-entries. Therefore, when the OpenFlow switch is targeted by a saturation attack, the TPR-value increases significantly, since a large portion of the incoming packets are table-miss packets. Thus, the TPR-value can be used as a feature, since it reflects the impact of the saturation attacks on the OpenFlow switches and provides insight into the nature of the processed packets.

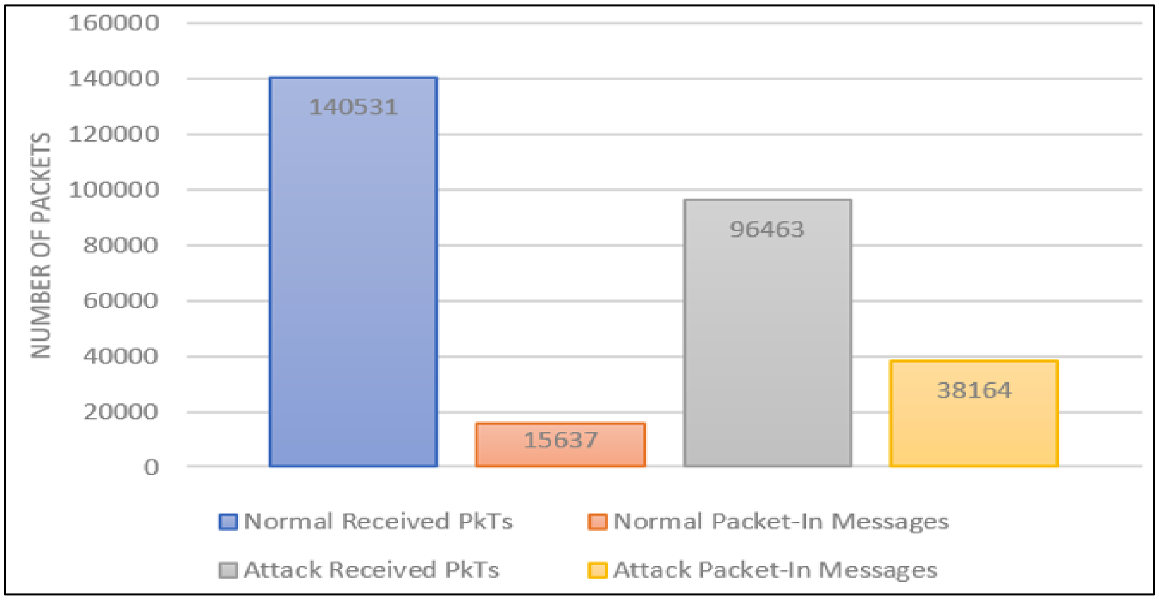


Figure 20 A Sample of Normal and Malicious Received Packets with Corresponding Malicious Table-Miss Packets for One Minute

4.2.1 Data Collection and Experiment Setup

Extracting the training datasets is a prerequisite for training the machine-learning classifiers. Thus, we have extracted datasets with time windows equal to one minute of OpenFlow traffic analysis from the captured benign and malicious OpenFlow traffic of the simulated SDN environments, as detailed in Chapter 3. The dataset features of this method are the entropy values of the source IPv4 addresses of the switch table-miss packets and the TPR-values. Similar to the first method, we evaluated the effectiveness of this method in detecting the targeted OpenFlow switches by known saturation attacks by adopting the K-NN, SVM, and NB classifiers. Also, we evaluated the degree of consistency between the victim switch detection models and the detection method model.

4.2.2 Experimental Results

As shown in Table 8, the detection performance of the supervised classifiers is highly improved by using this method. Also, our victim switch detection classifier was fully consistent with the detection method in all 31 tests. The victim switch detection using OpenFlow messages payload achieved a higher detection result in detecting the targeted OpenFlow switch than the previous method (see Section 4.1).

Table 8 Detection Results Using Entropy and TPR-Value

Algorithm	Precision	Recall	F1-Score
K-NN	96%	94%	95%
SVM	81%	85%	83%
NB	86%	90%	88%

The reason behind the improvement of the detection performance of the classifiers is related to the extracted features of this method. The entropy value of the table-miss IPv4 addresses and the TPR-value provide a clear distinction to the trained models to identify the targeted OpenFlow switches. Fundamentally, when an attacker tries to flood an OpenFlow switch, he or she sends a large number of packets with spoofed IPv4 address to urge the targeted OpenFlow switch to generate a large number of Packet-In messages.

Therefore, the entropy value can provide a clear distinction between the nature of the malicious Packet-In messages and the normal ones. Even in a large scale SDN environment, where a large amount of normal traffic is generated, the entropy value can be used to indicate the targeted OpenFlow switches. Because in normal traffic, the incoming flow packets have the same IPv4 address. Thus, when the OpenFlow switch processes the first packet of the incoming new traffic, a flow-rule will be installed and match the remaining packets. As a result, the entropy value of the IPv4 address of the table-miss packet of the normal traffic will be a small value. However, in malicious traffic, the incoming traffic entropy value of IPv4 addresses will be high, because all the incoming new packets IPv4 addresses are spoofed.

Also, the TPR-value is an effective feature which can be used to identify the targeted OpenFlow switches, because it reflects the nature of the processed traffic. For example, when an OpenFlow switch processes the network traffic and most of the packets of this traffic cause a table-miss, the TPR-value will be high, which indicates that the incoming traffic is suspicious. However, in normal traffic, the TPR-value will be low even in a large scale SDN network, because the number of table-miss packets with corresponding incoming traffic will be low, since most of the incoming traffic are benign packets and match the flow-rules of the connected OpenFlow switches.

The victim switch detection using the OpenFlow messages payload method by using entropy and the table-miss packets rate value (TPR-value) requires extra computation for extracting the IPv4 addresses from the Packet-In messages data field and calculating the relevant values for each OpenFlow switch in the SDN environment. Thus, it might cause a slight performance overhead over the SDN environment.

4.3 Victim Switch Detection Through Integration of OpenFlow Message Headers and Payload

This method integrates the features of the previous two methods as explained in sections 4.1 and 4.2. Therefore, the victim switch detection method will use the OpenFlow message headers to extract the number of Packet-In, Packet-Out, Packet-Mod, and TCP-ACK message features. It will use the Packet-In message data fields to extract the entropy value of the source IPv4 addresses of the switch table-miss packets, and also the TPR-value of each OpenFlow switch in the SDN environment. Thus, the number of Packet-In, Packet-Out, Packet-Mod, TCP-ACK messages, the entropy value of the source IPv4 addresses of the switch table-miss packets, and the TPR-value are used as machine learning classifier features to detect and identify the victim OpenFlow switches.

4.3.1 Data Collection and Experiment Setup

Similar to the previous two methods, the training and testing datasets have been extracted from the collected OpenFlow traffic. The time window of the extracted dataset is equal to one minute of OpenFlow traffic analysis. Also, we have conducted 31 tests to evaluate the degree of consistency between the victim switch detection models and the saturation attack detection method models.

4.3.2 Experiment Results

As shown in Table 9, the detection performance of the supervised classifiers obtained higher detection results than the previous two methods. Also, the K-NN classifier was fully consistent with the detection method in all 31 tests. We believe that combining the features of the previous two methods improves the detection performance of the classifiers in identifying the targeted OpenFlow switches, because the combined features reflect the impact of saturation attacks on the targeted OpenFlow switches and the OpenFlow traffic, which enable the trained classifiers models to accurately identify the targeted OpenFlow switches. As a result, this method was used in the online victim switch detection module to identify the targeted OpenFlow switches by known saturation attacks accurately.

Table 9 Detection Results Using the Integration Method

Algorithm	Precision	Recall	F1-Score
K-NN	95%	96%	95%
SVM	83%	90%	86%
NB	91%	88%	89%

4.4 Detecting OpenFlow Switches Targeted by Unknown Saturation Attacks

Machine learning detection systems are deficient in their ability to detect new attacks, which the classifier has not yet ‘seen’ (in the training phase) [51]. This is because the fundamental approach of a machine learning detection system is to train the classifier on examples of all available attack classes – typically, a large number of specimens for each attack class. It is difficult to find a dataset that includes an example of each attack class. Thus, when an unknown attack targets the network, the machine learning model fails to detect the attack, because there was no prior opportunity to train the model on this type of attack.

In our experiments, we extracted dataset features and then excluded the targeted attack and its combination of samples from the training dataset, in order to present it as an unknown attack to the trained model. In this case, the training dataset included benign traffic samples, as well as the remaining attacks and their respective samples. The testing dataset consisted of the unknown attack samples, as well as randomly selected benign traffic samples. For example, given that X is the training dataset that consists of attack samples and normal traffic samples, Y is the testing dataset that includes attack samples and normal traffic samples, G is the unknown attack samples only, and C is the attack combination samples, the X training datasets and the Y testing datasets are in the form of:

$$X_{(\text{trainingSet})} = X - G - C \tag{4.3}$$

$$Y_{(\text{testingSet})} = G + NormalTrafficSamples \tag{4.4}$$

Based on the reported results in Table 10, the supervised classifiers are capable of detecting the victim OpenFlow switches when it is targeted by unknown saturation attacks. Specifically, the K-NN classifier shows capable detection results. We hypothesize that all the different attacks in the training set allow the classifier to generalize the missing one.

Table 10 Detecting OpenFlow Switches Targeted by Unknown Attacks Using Supervised Classifiers

	K-NN			**SVM**			**NB**		
Attack	**P* %**	**R* %**	**F1* %**	**P* %**	**R* %**	**F1* %**	**P* %**	**R* %**	**F1* %**
UDP	100	94	97	99	90	94	99	92	95
SYN	100	92	96	100	48	65	99	98	98
TCP-SARFU	97	95	96	100	61	76	99	94	96
IP-Spoofing	97	94	95	98	65	78	91	93	92
ICMP	100	54	70	99	47	64	100	53	69

P = Precision, R* = Recall, and F1* = F1 score*

To confirm this hypothesis, we considered the K-NN (our best choice) where only one kind of attack is used in the training set. The results in Table 11 show that the K-NN in the worst-case scenario obtained a low detection performance result in identifying the victim OpenFlow switches when they are targeted by unknown saturation attacks. This means that supervised classification cannot be trusted for unknown SDN attacks in general.

For these reasons, we consider semi-supervised classifiers such as Isolation Forest, One class-SVM, Autoencoder, and Variational Autoencoders. In this case, the training set comprises only the normal instances, whereas, the testing set consists of all the normal and attack instances.

Table 11 K-NN Detection Results of Identifying Targeted OpenFlow Switches by Unknown Attacks

Attack	**K-NN**		
	Precision	**Recall**	**F1-Score**
UDP	0.96	0.40	0.56
SYN	0.93	0.17	0.29
TCP-SARFU	0.91	0.30	0.45
IP-Spoofing	0.94	0.23	0.18
ICMP	0.93	0.43	0.29

Table 12 reports the average results of the 10-fold cross-validation. The semi-supervised algorithms have higher detection performance results for identifying the victim OpenFlow switches than the supervised ones. Specifically, the Variational Autoencoder is effective in identifying the victim OpenFlow switches when they are targeted by unknown saturation attacks and obtains comparable results to the supervised classifiers in detecting the known saturation attacks. Therefore, our detection victim switch module adopted the Variational Autoencoder algorithm to identify the OpenFlow switches when they are targeted by known and unknown saturation attacks. However, other machine learning algorithms can be utilized in this module.

In this approach, the online victim switch detection module is a two-stage process (i.e., training stage and detection stages). In the training stage, before the system starts for the first time, the Variational Autoencoder is trained using a training dataset made in advance. At the detection stage, when our system is running, the OpenFlow traffic collector and feature extractor module send the extracted features (see section 4.3) periodically as instances for each OpenFlow switch in the SDN network.

Upon receiving these instances from the victim switch detection module, the constructed Variational Autoencoder model processes each of them to identify the OpenFlow switches under a saturation attack. The classification result of "1" indicates that the OpenFlow switch is under a saturation attack.

Table12 Semi-Supervised Algorithms Detection Results of Identifying Targeted OpenFlow Switches by Unknown Attacks

Algorithm	Precision (%)	Recall (%)	F1-Score (%)
Variational Autoencoder	93	98	96
Basic Autoencoder	84	81	82
One-Class SVM	73	75	74
Isolation Forest	82	67	73

4.5 Summary

In this chapter, we investigated three methods for identifying the targeted OpenFlow switches by known and unknown saturation attacks in an SDN network, incorporating both supervised and semi-supervised machine learning algorithms.

Based on the reported results, detecting the targeted OpenFlow switches by using the OpenFlow message headers (see section 4.1) did not provide very high precision, recall, and F1-score results in identifying the targeted OpenFlow switches due to the kind of features extracted from the OpenFlow message headers. Thus, using the OpenFlow message headers to determine the targeted OpenFlow switches is not a suitable approach, since the headers of these messages have a large overlap between the normal and malicious ones.

The experiment results showed that using the OpenFlow message payloads (see section 4.2) provides a higher detection performance in terms of precision, recall, and F1-score since the extracted features accurately reflect the behavior of the OpenFlow switches when they are under a saturation attack. This enables the trained models to identify these switches accurately. Finally, the integration of OpenFlow message headers and payload (see section 4.3) obtained the highest detection results. This was because the features extracted from the OpenFlow message headers and payloads allowed the trained model to precisely identify the OpenFlow switches targeted by known and unknown saturation attacks.

CHAPTER FIVE: COUNTERMEASURE SATURATION ATTACKS

The countermeasure module triggers when the victim switch detection module identifies the OpenFlow switches that are targeted by saturation attacks, otherwise, it remains idle. It is an efficient and cost-effective countermeasure method that does not require any modification of the SDN design or any extra device. It can mitigate a family of saturation and hybrid saturation attacks by utilizing three components. (1) The Packet-In deep inspection filter is responsible for identifying the zombie hosts, targeted hosts, and reducing the false-positive rate of the victim switch detection module. (2) The blocking-rule manager component is responsible for blocking the malicious incoming traffic from the zombie hosts. Lastly, (3) the flow-rule manager component can accurately identify the installed malicious flow-entries and remove them from the flow-tables of the victim OpenFlow switches.

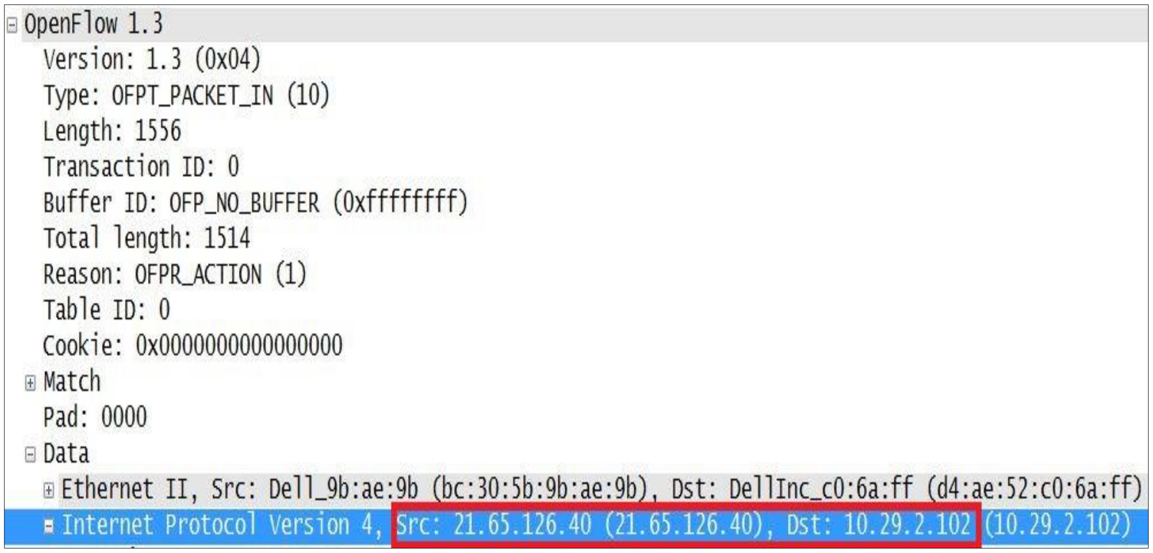
In the upcoming sections, we first introduce the Packet-In deep inspection filter. We continue by discussing the mitigation of known and unknown saturation attacks by describing the blocking-rule manager and flow-rule manager. We describe the implementation of the proposed defense system and the setup of our experiments. Finally, we present the proposed defense system experimental results.

5.1 Packet-In Deep Inspection Filter

The Packet-In deep inspection filter can identify the zombie hosts, the targeted destination, and reduce the false-positive rate of the victim switch detection module by inspecting the Packet-In messages of each OpenFlow switch identified as a victim by the victim switch detection module. Algorithm 2 shows the working process of the Packet-In deep inspection filter, as follows:

- Extract all the Packet-In messages of each OpenFlow switch that has been identified as a victim by the victim switch detection module (lines 1-2).
- Inspect the data field for each Packet-In message. The data field contains the header of the table-miss packet or the whole table-miss packet, as depicted in Figure 21. Through the inspection, the source and the destination IPv4 addresses, MAC addresses, and switch port numbers of the table-miss packets are extracted (lines 3-7).
- Identify the zombie hosts and the target destinations by comparing the table-miss packets source and destination IPv4 addresses with the saved network topology that has been obtained by the network topology manager module. Essentially, the attacker keeps spoofing the content of the transmitted packets to reduce the possibility of matching any flow-rules. This is done in order to urge the victim switch to generate Packet-In messages – specifically, the source IPv4 address of the table-miss packets (lines 8-14). The malicious packets with the same IPv4 address will drastically downgrade the performance of the saturation attacks since the controller will install the flow-rule on the OpenFlow switch flow table that matches the incoming traffic. Therefore, the zombie hosts can be identified by comparing the MAC address, port number, and source IPv4 address of the table-miss packet with the saved network topology. If the source IPv4 address has zero matches with the network topology and the MAC address,

and/or the port number matches the saved network topology, then the table-miss packet is considered a malicious packet and the host with the corresponding source MAC address is recognized as a zombie host. Also, the destination of the table-miss packet is regarded as a targeted destination and the OpenFlow switch of the Packet-In messages will be regarded as a victim switch.



```
⊟ OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_PACKET_IN (10)
    Length: 1556
    Transaction ID: 0
    Buffer ID: OFP_NO_BUFFER (0xffffffff)
    Total length: 1514
    Reason: OFPR_ACTION (1)
    Table ID: 0
    Cookie: 0x0000000000000000
  ⊞ Match
    Pad: 0000
  ⊟ Data
    ⊞ Ethernet II, Src: Dell_9b:ae:9b (bc:30:5b:9b:ae:9b), Dst: DellInc_c0:6a:ff (d4:ae:52:c0:6a:ff)
    ⊟ Internet Protocol Version 4, Src: 21.65.126.40 (21.65.126.40), Dst: 10.29.2.102 (10.29.2.102)
```

Figure 21 A Sample of a Table-Miss Packet Data Field

Algorithm 2: Packet-In Deep Inspection Filter

Input	*victim OpenFlow Switches OFSwitches, OpenFlow traffic OF, Network Topology NT*
Output	*ZombiesHostMACs, ZombiesHostIPs, TargetDestinationIP*
Declare	*vSwitch, vSwitch_PacketIn, sourceIP, sourceMAC, destinationIP*
Steps	

```
1   For (i = 0, i <= OFSwitches, i++ ) do
2       vSwitch = OFSwitches[i]
3       vSwitch_PacketIn = extractPacketInMessages(vSwitch, OF)
4       For each packet-In  in  vSwitch_PacketIn do:
5           sourceIP = extractSourceIPaddressFromDataField(packet-In  )
6           sourceMAC = extractSourceMACaddressFromDataField (packet-In  )
7           destinationIP = extractdestinationIPaddressFromDataField (packet-In  )
8           If (sourceIP not in NT ) {
9               ZombieHostsIPs ← [vSwitch , sourceIP]
10              ZombieHostsMACs ← [vSwitch , sourceMAC]
11              TargertDestinationIP ← destinationIP
12          End if
13      End for
14  End for
```

5.2 Blocking Rule Manager

The blocking-rule manager is triggered by the Packet-In deep inspection filter. It is responsible for mitigating saturation attacks by blocking the malicious incoming traffic from the zombie hosts. After the Packet-In message filter identifies the zombie hosts, the blocking-rule manager obtains the OpenFlow switches of the zombie hosts by comparing their MAC addresses and port numbers with the saved network topology. Next, it installs a high priority blocking-flow rule on these zombie hosts' OpenFlow switches using controller-to-switch messages.

As shown in Figure 22, the blocking- rule consists of a switch DPID field which is equal to the zombie host's OpenFlow switch DPID. The priority field is used to assign the priority of the installed flow-rule since the processing of the flow-rules is based on priority. Thus, the assigned value should be the highest, which is equal to 32,767. The MAC address field is equal to the zombie host's MAC address; the ingress port field is equal to the zombie host's OpenFlow switch port; and finally, the action field is equal to the "*Drop*" action to block the malicious incoming traffic from the zombie hosts. After installing the blocking-rule, the incoming malicious traffic from the ingress port will be matched against the blocking-rule. Once the blocking-rule manager installs the blocking entry, the malicious incoming traffic will be blocked. Subsequently, the flow-rule manager will be triggered.

Switch DPID	**Priority**	**MAC Address**	**Port Number**	**Action**
00.00.00.03	32767	Bc: 30:5b:9b:ae:9b	1	Drop

Figure 22 A Sample Blocking Flow Rule

5.3 Flow Rule Manager

One of the main destructive consequences of a saturation attack is preventing legitimate flow-entries from being installed. This happens because the attack consumes the data plane memory by installing a huge amount of malicious flow-entries into the victim switches' flow-tables. Therefore, identifying the malicious flow-entries and removing them from the victim switch's flow-tables is an important step in resisting saturation attacks.

As far as we know, few studies have been proposed to tackle this issue. The basic method proposed in [71] is to delete all the flow entries from the victim OpenFlow switch, which sacrifices the legitimate flow-entries. Another approach proposed in [72] is to keep track of any modification to the OpenFlow switch's flow-tables by recording any addition or deletion of the switch's flow-entries. This approach can cause performance overhead for the SDN environment. The proper solution should identify malicious flow-entries accurately without removing the legitimate ones.

Identifying malicious flow-entries is an issue [73]. When a table-miss occurs, the controller installs flow-entries into the switch's flow-tables to match the new incoming traffic. The process of creating flow-entries does not exhibit a single behavior. The controller may use the port numbers, switch DPID, and/or MAC addresses of the source and destination, IPv4 addresses of the source and destination, or their combinations to match the incoming traffic. Therefore, there are different forms of malicious flow-entries.

The flow-rule manager component can accurately identify the malicious flow-rules and remove them from the victim OpenFlow switches flow tables. Figure 23 shows the working process of the flow-rule manager, divided into two phases:

- In the first phase, the flow-rule manager creates the saturation attack topology by obtaining all the source and destination IPv4 addresses, MAC address, victim OpenFlow switch DPIDs, and port numbers of the victim OpenFlow switch's table-miss packets.
- In the second phase, the flow-rule manager obtains the victim OpenFlow switch's flow-rules by using a controller-to-switch message. Subsequently, it compares the values of the flow-rules fields with the attack topology. If

any value of the flow-entry fields matches any of the saturation attack topology values, the flow-entry will be considered malicious. In this case, the flow-rule manager uses the controller-to-switch message to delete the identified malicious flow-entries from the flow-tables of the victim OpenFlow switch. As a result, a large amount of memory is freed on the OpenFlow switch flow-tables, which allows the new legitimate flow-entries to be installed and returns the OpenFlow switch settings to their pre-attack state.

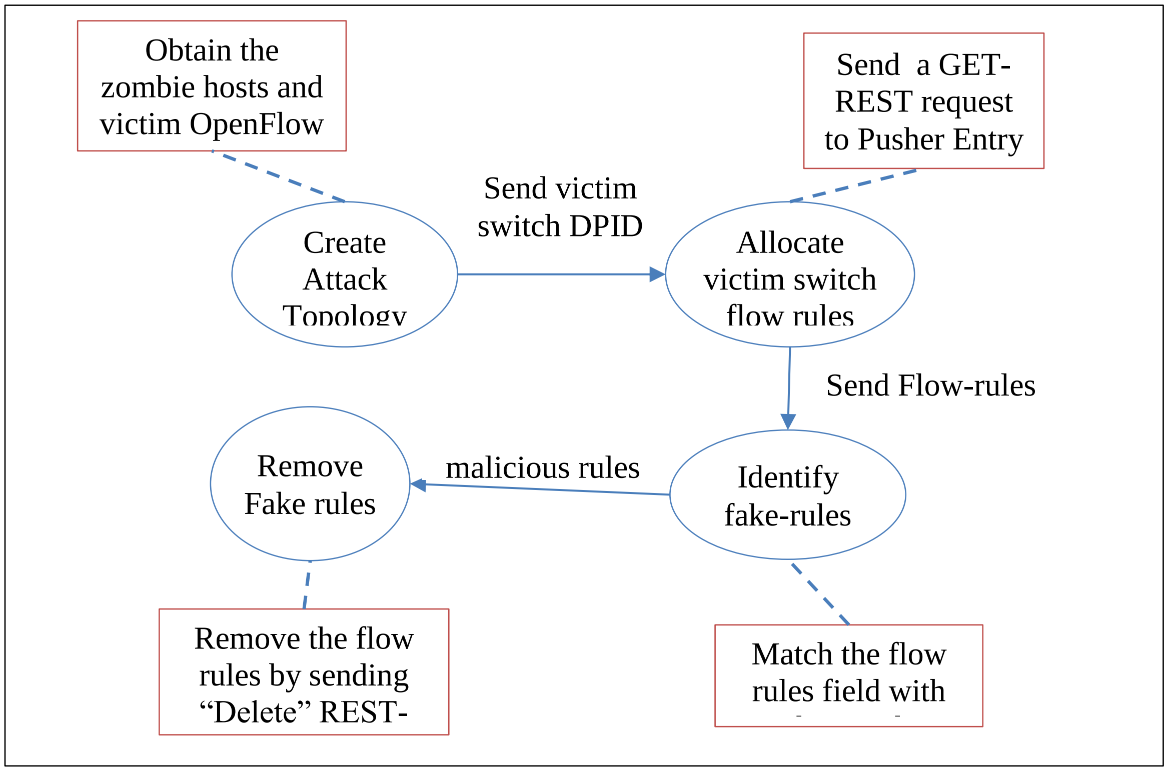

Obtain the zombie hosts and victim OpenFlow
Send a GET-REST request to Pusher Entry
Send victim switch DPID
Create Attack Topology
Allocate victim switch flow rules
Send Flow-rules
Remove Fake rules
malicious rules
Identify fake-rules
Remove the flow rules by sending "Delete" REST-
Match the flow rules field with

Figure 23 The Flow-Rule Manager State Machine

5.4 System Implementation

We implemented the proposed defense system, including the network topology manager, traffic collector and feature extractor, saturation attack detection, victim switch detection, Packet-In deep inspection filter, blocking-rule manager, and flow-rule manager. All of them are implemented as an application on Floodlight master V1.2 [74] in Python. Meanwhile, we installed and used the Mininet tool to create simulated SDN environments with different network topologies and scales on a computer equipped with an i5 CPU and 8 GB of RAM.

To compare the proposed defense system with previous work, we launched the saturation attacks in two scenarios: (1) an SDN network without any protection, and (2) SDN networks with the protection of the proposed defense system.

5.4.1 The Network Topology Manager Module

The network topology manager module is responsible for obtaining the SDN network topology by incorporating two elements. First, the network topology extractor component uses the northbound APIs exposed by the controller to obtain the network topology. It uses the REST API to communicate with the Topology Manager / Routing service that maintains the current network topology information. This information consists of the IPv4, IPv6, and MAC addresses, and port numbers for the connected switches, hosts, and controllers.

Second, the network topology analyzer is responsible for analyzing and parsing the obtained network topology information. First, the topology analyzer extracts the connected OpenFlow switches' DPIDs addresses, MAC addresses. Second, for each OpenFlow switch, it extracts the connected hosts' IPv4 address, MAC addresses, and port numbers. At this point, the traffic collector and feature extractor module are initiated.

5.4.2 The Traffic Collector and Feature Extractor Module

The objectives of the traffic collector and the feature extractor module are to collect the OpenFlow traffic between the controller and the OpenFlow switches, in order to extract the saturation attack detection module and victim switch detection module features. This module consists of the traffic collector and the feature extractor.

5.4.2.1 OpenFlow Traffic Collector Component

In SDN environments, there are two common methods to collect the OpenFlow traffic between the control plane and the data plane. The first method is to use the controller-to-switch flow statistics messages. The controller can periodically send these messages and the OpenFlow switch responds with one or more reply messages containing the flow statistics. However, when the SDN environment is under a saturation attack, the OpenFlow channel bandwidth is congested by a large number of table-miss packets that fill the flow-tables of the switches, which makes it hard for the OpenFlow switches to reply in a timely manner. Also, the flow statistics messages are large messages, which may congest the OpenFlow channel even further [75].

Since the controller may have been exhausted from processing the malicious table-miss packets that are coming from the OpenFlow switches, it may not receive, process, and respond to the flow statistics messages in a timely manner. For these reasons, using the flow-statistics is not a reliable option for a saturation attack detection system.

The second common method that has been used by different research studies is sFlow-RT [76], which is software that can be installed in the SDN environment to collect and monitor the OpenFlow traffic. However, sFlow-RT uses a periodic sampling of the OpenFlow traffic and cannot collect information on all OpenFlow packets, which may have a large impact on the accuracy of the detection method [77]. In addition, sFlow-RT cannot collect a low-rate of OpenFlow traffic [78] [79].

To overcome the shortcomings of the two previous methods for collecting the OpenFlow traffic, we adopted the Pyshark library [80] in the proposed defense approach. Pyshark is a wrapper package of the tshark library [81] that provides a live capture of OpenFlow traffic and easy parsing of the collected traffic packets. More importantly, Pyshark can provide full insight into each OpenFlow packet that is transferred between the control plane and the data plane, without consuming the OpenFlow channel bandwidth. Therefore, the traffic collector uses Pyshark as the OpenFlow traffic collection method.

The OpenFlow traffic collection process is a session-based process and the collected OpenFlow traffic of each session is fed to the feature extractor. The duration of each session is equal to the pre-defined time window of OpenFlow traffic analysis. In this approach, the OpenFlow traffic time window is set to equal one minute, based on the identification of the proper time window for OpenFlow traffic analysis, as described in Chapter Three.

5.4.2.2 Feature Extractor Component

From the collected OpenFlow traffic, the feature extractor component parses the OpenFlow traffic, extracts features, and feeds them as an instance to the saturation attack detection module to determine if there is abnormal behavior in the SDN network. In this approach, four features are extracted from the OpenFlow traffic: (1) the number of Packet-In messages sent from the switches to the controller, (2) the number of Packet-Out messages sent from the controller to the switches, (3) the number of Packet-Mod messages sent from the controller to the switches, and (4) the number of TCP-ACK messages sent from the switches to the controller.

When the saturation attack detection module discovers abnormal behavior, the feature extractor processes the collected OpenFlow traffic to extract the features for the victim switch detection module. The features used by the victim switch detection module are: (1) the number of Packet-In messages, (2) the number of Packet-Out message, (3) the number of Packet-Mod messages, (4) the number of TCP-ACK messages, (5) the entropy values of the source IPv4 table-miss packets, and (6) the TPR-value for each OpenFlow switch in the SDN network.

5.4.3 Saturation Attack Detection Module

The saturation attack detection module is utilized to detect saturation attack on SDN network. It employs the Variational Autoencoder classifier as its main machine learning algorithm. Essentially, the Variational Autoencoder is a kind of semi-supervised learning algorithm that requires a smaller training dataset, compared to supervised machine learning classifiers, and is effective at detecting the known and unknown saturation attacks.

In this work, the saturation attack detection module is a two-stage process (i.e., training stage and detecting stage). In the training stage, before running the defense system, the Variational Autoencoder is trained using the training dataset, which was made in advance. As discussed in Chapter 3, the extracted number of Packet-In, Packet-Out, Packet-Mod, and TCP-ACK messages are utilized as the features extracted from the collected OpenFlow traffic. As shown in Table 13, the training dataset consists of 104,512 attack and normal samples.

Subsequently, in the detection stage, after running this defense system, the OpenFlow traffic collector and feature extractor module extract the features from the real-time captured OpenFlow traffic and provides them as an instance to the Variational Autoencoder, to determine if there is abnormal behavior in the SND environment

Table 13 Saturation Attack Detection Module Training Dataset

Sample Type	Number of samples
Attack Sample	8,963
Normal Sample	95,549
Total	**104,512**

5.4.4 Victim Switch Detection Module

The victim switch detection module utilizes the Variational Autoencoder algorithm due to its effectiveness at detecting the OpenFlow switches targeted by known and unknown saturation attacks, as reported in Chapter 4. Similar to the saturation attack detection module, the victim switch detection module is a two-stage process, the training stage and the detection stage. In the training stage, the Variational Autoencoder is trained using the pre-made training dataset. As described in Chapter 4, the number of Packet-In, Packet-Out, Packet-Mod, TCP-ACK messages, the entropy value of the IPv4 addresses of the table-miss packets, and the TPR value are features of the training dataset. As depicted in Table 14, the training dataset includes 160,000 samples of normal and attack traffic.

Next, in the detection stage, when the system is running and after the saturation attack detection module discovers abnormal behavior, the OpenFlow traffic collector and feature extractor module extract the victim switch detection module features from the real-time captured OpenFlow traffic. These are provided as instances to the constructed

Variational Autoencoder model, one for each OpenFlow switch in the SDN network, in order to identify the OpenFlow switches targeted by known and unknown saturation attacks.

Table 14 Victim Switch Detection Module Training Dataset

Sample Type	**Number of samples**
Attack Sample	65,000
Normal Samples	95,000
Total	**160,000**

5.4.5 Countermeasure Module

The countermeasure module can mitigate the saturation attacks by blocking the incoming attack traffic and remove the installed fake flow rules in the OpenFlow victim switches flow tables during the attacks. It consists of the blocking-rule manager and the flow-rule manager that utilize the REST-API services provided by the controller northbound APIs.

The blocking-rule manager creates a request message as a JSON object that consists of the blocking rule(s) and sends it over HTTP to the Flow-Entry Pusher service on the controller side. Subsequently, the Flow-Entry Pusher installs the blocking rule(s) into the OpenFlow switches' flow-tables to block the incoming malicious traffic from the zombie host(s). Later, a confirmation message is sent by the Flow-Entry Pusher to the blocking-rule manager to confirm the installation of the blocking rules.

The Flow-rule manager creates an attack topology that includes all the spoofed IPv4 addresses, MAC addresses of the zombie hosts and targets hosts, the DPIDs of the zombie hosts' OpenFlow switches, and the DPIDs of the victim OpenFlow switches. Next, it creates a JSON request message for obtaining the flow-rules of the victim OpenFlow switches. Upon receiving the request messages, the Flow-Entry Pusher obtains the flow-rules of all OpenFlow switches listed in the request message and sends them as a response message over HTTP to the flow-rules manager. Once the response message is received by the flow-rule manager, it matches the received flow-rules against the attack topology in order to identify the malicious ones. Next, another request message (or messages) will be created which contains all the fake rules that should be removed from the flow-tables of the victim OpenFlow switches by the flow-rule manager. Upon receiving these messages from the Flow-Pusher Entry, the listed flow-rules in these messages will be deleted. Subsequently, the Flow-Pusher Entry will send a confirmation message to the flow-rule manager to confirm that the fake flow rules have been eliminated.

5.5 Setup of Experiments

First, the saturation attack detection module and the victim switch detection module need to be trained in advance. Before running the system, both of them were trained using pre-made training datasets, as explained in the previous section.

Second, the precision, recall, and F-1 score were used to evaluate the performance of the detection module and the victim switch detection module.

Third, by using our testing parameters as demonstrated in Table 15, we created a simulated SDN environment for each experiment. The environments have different network topology, network scale, number of targeted OpenFlow switches, number of zombie hosts, and types of saturation attacks. We conducted 31 experiments using the Hping3 tool that covers all the SYN flooding, UDP flooding, ICMP, IP Spoofing, and SARFU-TCP flooding attacks and their combinations.

Table 15 The Experiments' Testing Parameters

Testing Parameter	Description	Value
Tn	Network topology	Star, mesh, linear, tree
Ts	Network scale	Small, Medium, Large
At	Attacks type	SYN, UDP, ICMP, TCP-SAFRU, IP-Spoofing and their combination (31 attacks)
SWn	Number of victim switches	1 switch – ½ of the total number of switches
Hn	Number of zombie hosts	1 switch – ½ of the total number of hosts

Fourth, we evaluated the performance of the countermeasure module in mitigating the saturation attacks by measuring the CPU utilization of the controller via the NetData tool [82] and the bandwidth of the OpenFlow connection channel by using the IPref tool [83] before trigging the attack, during the attack, and after the attack was mitigated.

Fifth, we evaluated the effectiveness of our countermeasure module for identifying and removing the malicious flow-rules from the targeted OpenFlow victim switches by measuring the flow-table utilization of the victim switches under OpenFlow, with and without the protection of the proposed defense system.

Sixth, we compared the CPU utilization of the controller and the flow-tables utilization of the victim switch to the CPU utilization and flow-tables utilization of FDAM, FloodDefender, and FloodGuard. Unfortunately, due to copyright issues, we are not able to get the source code of FloodDefender, FDAM, and FloodGuard. Therefore, we used their published performance results to compare to the proposed defense system's performance.

5.6 Experimental Results

5.6.1 Detecting Saturation Attacks

The saturation attack detection module was capable of detecting the known and unknown saturation attacks in each experiment with 85% precision, 97% recall, and a 91% F1-score within a 0.2-second prediction time. Based on these results, the proposed defense system provides a higher detection performance than FDAM, FloodDefender, and FloodGuard. The FDAM system adopts the SVM classifier to detect the SYN, UDP, and ICMP flooding attacks. In contrast, the proposed defense system is capable of detecting TCP-SYN, ICMP, UDP, and IP-Spoofing attacks as well as the hybrid saturation attacks.

Also, the proposed defense system is capable of detecting unknown saturation attacks, whereas the FDAM is deficient in detecting these attacks. FloodGuard is designed to detect and countermeasure the SYN flooding attack only, while FloodDefender is developed to detect and mitigate UDP, SYN, and ICMP flooding attacks. However, these systems are not capable of detecting hybrid saturation attacks and unknown attacks.

5.6.2 Victim Switch Identification

In each experiment, the victim switch detection module was able to identify the OpenFlow switches targeted by known and unknown saturation attacks. The average precision was 93%, the average recall was 98%, and the average F-1 score was 96%. The predication time of the Variational Autoencoder was equal to 0.9 seconds. Based on these results, the victim switch detection module showed high performance at detecting and identifying the targeted OpenFlow switches.

5.6.3 Computational Resources Utilization

Figure 24 shows the protection of the computational resources of the SDN environment. Before the attack occurred (from the 0th second to the 58th second), the average CPU utilization was about 45-50%, since the simulated SDN environment and the controller were running on the same machine. When the attack occurred (from second 59 to 119), the CPU utilization reached around 95%. Meanwhile, the OpenFlow traffic collector and feature extractor collected the OpenFlow traffic and extracted the victim switch detection module features. Then at second 120, the CPU utilization went down quickly (to around 45%) because our system identified the targeted OpenFlow switches and mitigated the saturation attack. The total execution time of the victim switch detection and countermeasure modules was about 2.36 seconds.

Also, we have measured the bandwidth of the OpenFlow connection channel before, during, and after the mitigation of the attack, as depicted in Figure 25. The average bandwidth of the OpenFlow connection channel before the attack was about 3.2 GBPS. When the attack occurred, its bandwidth dramatically decreased to 0.43 GBPS. After our system mitigated the saturation attack, the OpenFlow connection channel bandwidth started increasing slowly, due to the huge amount of Packet-Out and Packet-Mod messages forwarded from the controller to the victim OpenFlow switch.

The results show that the proposed defense system saves the SDN environment's computational resources effectively without any noticeable performance overhead. It operates without the need to add extra devices, as in FloodGuard, or flood the neighbors' OpenFlow switches, as in FloodDefender.

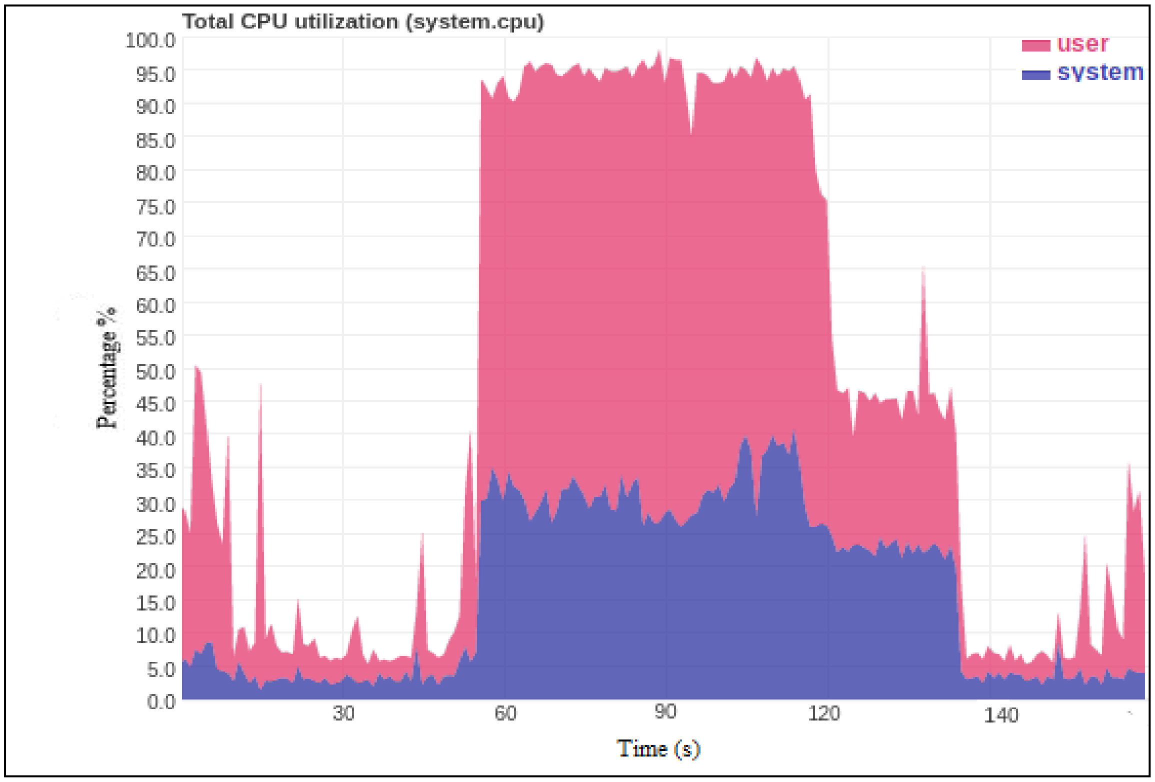


Figure 24 CPU Utilization Under a UDP Saturation Attack

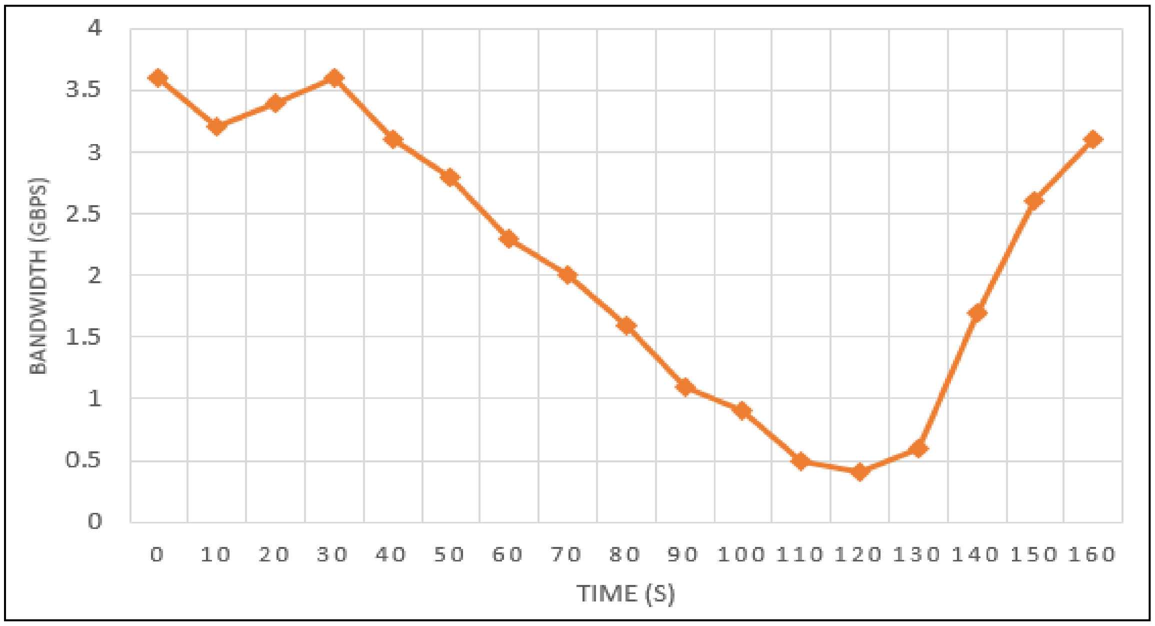


Figure 25 OpenFlow Connection Channel Under a UDP Saturation Attack

5.6.4 Flow Table Utilization

The flow-table utilization of the victim OpenFlow switches is depicted in Table 16. We find that the proposed defense system did not overload the network when there is no saturation attack. When the attack occurred, the flow-table utilization of the victim OpenFlow switch without any protection reached 100% because of the installation of the malicious flow-rules. With the protection of the proposed defense system, the flow-table utilization remained steady, since all the malicious flow-rules were removed from the victim OpenFlow Switches.

Also, we observe that the total flow-table utilization rate caused by the proposed defense system is no greater than 1% because of the installation of the blocking flow rules. With FloodDefender, the flow-table utilization can reach up to 15% of the flow-table buffer, due to the installation of monitoring and processing flow-rules. With FloodGuard, the flow-table utilization can reach up to 30%, since it uses rate control to protect the controller and OpenFlow switches. Therefore, the proposed defense system provides a more efficient way of handling malicious table-miss packets with lower flow-table utilization.

Table 16 Flow-Table Utilization Under a UDP Saturation Attack

	OpenFlow	**Our System**	**FloodDefender**	**FloodGuard**
No Attack	4% ~ 5%	4% ~ 5%	4% ~ 5%	4% ~ 5%
Under Attack	100%	5% ~ 6%	19% ~ 20%	34% ~ 35%

5.7 Summary

This chapter introduced the countermeasure module which is responsible for mitigating saturation attacks and eliminating their consequences, by utilizing the Packet-

In deep inspection filter, the Blocking-rule manager, and the Flow-rule manager. Also, it demonstrated the implementation of the proposed defense system and the experimental results.

Based on the reported results, the proposed defense system can protect the computation resources of the control plane and the bandwidth of the OpenFlow connection channel. Additionally, it can improve the flow-table utilization of the data plane without the need for extra devices and with very minimal resource consumption.

CHAPTER SIX: CONCLUSION

6.1 Summary

This dissertation introduced a deployable and effective defense framework against SDN saturation attacks. The proposed defense system can protect the control plane, data plane, and OpenFlow connection channel against the known and unknown saturation attacks.

The proposed defense system consists of: (1) a saturation attack detection module to detect the known and unknown saturation attacks in the early stages, (2) a victim switch detection module that can identify the OpenFlow switches targeted by known and unknown saturation attacks, and (3) a countermeasure module that is capable of mitigating these attacks and removing their consequences from the victim OpenFlow switches.

During the design of the proposed defense system, we studied the impact of different time-windows of OpenFlow traffic analysis on the detection performance of supervised machine learning classifiers in detecting the known saturation attacks. Also, we investigated the detection performance of supervised and semi-supervised classifiers in detecting the unknown saturation attacks in order to identify the most appropriate saturation attack detection method. Based on reported results in chapter 3, a slight variation of the time-window of OpenFlow traffic analysis has an obvious impact on the detection performance of the supervised classifiers. Also, the experimental results showed that supervised classifiers are not effective in detecting unknown saturation attacks. In contrast, the semi-supervised classifiers have the capability to detect the unknown saturation attacks effectively, specifically, the Variational Autoencoder algorithm (see chapter 3).

Moreover, we studied three victim switch detection methods to detect the OpenFlow switches targeted by known and unknown saturation attacks with the integration of supervised and semi-supervised classifiers to discover the most effective one. The experimental results showed that (see Chapter 4), the best performance was achieved by the Variational Autoencoder machine learning algorithm used in combination with the "Victim Switch Detection Through the Integration of OpenFlow Messages Headers and Payload" method (see sections 4.3 and 4.4). This combination accurately identified the OpenFlow switches targeted by known and unknown saturation attacks with 93% precision, 98% recall, and a 96% F1-score.

Furthermore, we studied different mitigation approaches to provide the most efficient and scalable countermeasure method against these attacks without adding new devices or changing the design of the SDN architecture. Based on the reported results in Chapter 5, our countermeasure method can protect the SDN environment against known and unknown saturation attacks.

We implemented and evaluated the performance of the proposed defense system by conducting extensive experiments that cover the TCP-SYN, UDP, IP-Spoofing, ICMP, and TCP-SARFU attacks and their combinations. In each experiment, a structured process was identified to create different SDN networks that mimic real-world SDN environments.

The reported results demonstrated that the proposed defense system is effective and efficient at detecting saturation attacks, identifying the targeted OpenFlow switches, and mitigating these attacks without causing overhead for the SDN environment.

6.2 Future Work

Our current work focuses on detection and countermeasure saturation network attacks against a single-controller SDN paradigm. For future work, we intend to focus on investigating different security aspects of SDN. Specifically, investigating the multi-controller SDN architecture security issues.

In recent days, multi-controller SDN architecture has been proposed to solve many security issues related to the single-controller SDN architecture, such as a single point of failure. The multi-controller SDN architecture can be divided into flat architecture and hierarchal architecture. In flat architecture, the SDN environment is divided into multiple domains in different locations, where, each domain is controlled by a controller and the controllers communicate with each other via east-bound interfaces. In hierarchal architecture, the controller layer is divided into two layers, the master layer, which consists of a master controller that is responsible for monitoring the entire network and the slave layer that includes many controllers that control many local domains.

The multi-controller SDN paradigm presents a major set of challenges. By adapting multi-controller SDN architecture, we cannot assure high reliability and availability of the SDN environment, since the attackers can target the connection links between the controllers, or the controllers could be overwhelmed by processing the malicious packets. Thus, the targeted controller(s) and the connected OpenFlow switches will be isolated from the entire network. Therefore, designing a multi-controller SDN defense system that can monitor the distributed controllers into multiple domains, detect the incoming attacks, and countermeasure them without affecting the availability of the entire SDN network is a crucial issue.

Another open issue is the scalability of multi-controller SDN architecture. In the multi-controller environment, the scalability relies on the number of controllers, the number of OpenFlow switches per controller, and the domain of the controller (i.e., controller location). Thus, if the number of OpenFlow switches per controller is irrationally assigned or the controllers deployed randomly into different domains, the performance of the entire network will be drastically affected. Thus, designing an approach/algorithm that can assign the right number of OpenFlow switches for each controller with the corresponding deployment domain for different network scale and topologies is another important issue.

Finally, the consistency of multi-controller SDN architecture is another challenging issue. In this architecture, the whole network is divided into multiple domains and each domain is controlled and managed by a controller. Therefore, the consistent and coherent information about the network is crucial to the multi-controller SDNs ability to make the right decision, such as installing flow-rules. Otherwise, out-synchronization between controllers or outdated network information could lead to unexpected behavior. Thus, developing an approach that provides high consistency and synchronization in multi-controller SDN architecture is another key issue.

The proposed defense system is capable of protecting single controller SDN environments against known and unknown saturation attacks. Our future work will focus on designing a defense system that is capable of detecting and mitigating saturation attacks in multi-controller SDNs.

REFERENCES

[1] https://www.ibm.com/services/business-continuity/sdn-versus-traditional-networking

[2] https://www.opennetworking.org/

[3] T. Mahjabin, Y. Xiao, G. Sun, and W. Jiang, “A survey of distributed denial-of-service attack, prevention, and mitigation techniques,” *International Journal of Distributed Sensor Networks*, vol. 13, no. 12, p. 155014771774146, 2017.

[4] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, “Network anomaly detection: methods, systems and tools,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[5] K. Johnson Singh, K. Thongam, and T. De, “Entropy-based application layer ddos attack detection using artificial neural networks,” *Entropy*, vol. 18, no. 10, p. 350, 2016.

[6] R. F. Fouladi, C. E. Kayatas, and E. Anarim, “Frequency based ddos attack detection approach using naive bayes classification,” in *2016 39th International Conference on Telecommunications and Signal Processing (TSP)*. IEEE, 2016, pp. 104–107.

[7] P. M. Mafra, V. Moll, J. da Silva Fraga, and A. O. Santin, “Octopus-iids: An anomaly based intelligent intrusion detection system,” in *The IEEE symposium on Computers and Communications*. IEEE, 2010, pp. 405–410.

[8] C. Wagner, J. François, T. Engel, “Machine learning approach for ip-flow record anomaly detection,” in *International Conference on Research in Networking*. Springer, 2011, pp. 28–39.

[9] Z. Muda, W. Yassin, M. Sulaiman, N. I. Udzir, “A k-means and naive bayes learning approach for better intrusion detection,” *Information technology journal*, vol. 10, no. 3, pp. 648–655, 2011.

[10] C. J. Hsieh and T. Y. Chan, "Detection ddos attacks based on neural-network using apache spark," in *2016 International Conference on Applied System Innovation (ICASI)*, IEEE,2016, pp. 1-4.

[11] M. H. Bhuyan, D. Bhattacharyya, and J. K. Kalita, "An effective unsupervised network anomaly detection method," in *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, ACM, 2012, pp. 533-539.

[12] X. Yuan, C. Li and X. Li, "Deepdefense: identifying ddos attack via deep learning," in *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, IEEE, 2017, pp.1-8.

[13] S. Yadav and S. Subramanian, "Detection of application layer ddos attack by feature learning using stacked autoencoder," in *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)*, IEEE, 2016, pp. 361-366.

[14] X. Qin, T. Xu and C. Wang, "Ddos attack detection using flow entropy and clustering technique," in *2015 11th International Conference on Computational Intelligence and Security (CIS)*, IEEE, 2015, pp. 412-415.

[15] R. M. Saad, M. Anbar, S. Manickam, and E. Alomari, "An intelligent icmpv6 ddos flooding-attack detection framework (v6iids) using back-propagation neural network," *IETE Technical Review*, vol.33, no. 3, pp. 244-255, 2016.

[16] C. Yin, Y. Zhu, J. Fei and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *Ieee Access*, vol. 5, pp. 21954-21961, 2017.

[17] N. Farnaaz and M. Jabbar, "Random forest modeling for network intrusion detection system," *Procedia Computer Science*, vol. 89, pp. 213-217, 2016.

[18] A. J. Malik, W. Shahzad, and F. A. Khan, "Network intrusion detection using hybrid binary pso and random forests algorithm," *Security and Communication Networks*, vol. 8, no. 16, pp. 2646-2660, 2015.

[19] J. David and C. Thomas, "Ddos attack detection using fast entropy approach on flow-based network traffic," *Procedia Computer Science*, vol. 50, pp. 30-36, 2015.

[20] N. Hoque, D. Bhattacharyya, and J. Kalita, "Denial of service attack detection using multivariate correlation analysis," in *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*, ACM, 2016, p. 100.

[21] M. Karakus and A. Durresi,"A survey: Control plane scalability issues and approaches in software-defined networking (sdn)," *Computer Networks*, vol. 112, pp. 279-293, 2017.

[22] D. Kreutz, F. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 55-60.

[23] R. Swami, M. Dave, and V. Ranga, "Software-defined networking-based ddos defense mechanisms," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, p. 28, 2019.

[24] J. Asharf and S. Latif, "Handling intrusion and ddos attacks in software defined networks using machine learning techniques," in *2014 National Software Engineering Conference*. IEEE, 2014, pp. 55-60.

[25] Q. Niyaz, W. Sun, and A. Javaid, "A deep learning based ddos detection system in software-defined networking (sdn)," *arXiv preprint arXiv:1611.07400*, 2016.

[26] A. A. Aizuddin, M. Atan, M. Norulazmi, M. M. Noor, S. Akimi, and Z. Abidin, "Dns amplification attack detection and mitigation via sflow with security-centric sdn," in *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication*. ACM, 2017, p. 3.

[27] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *LCN*, vol. 10, 2010, pp. 408-415.

[28] T. A. Tang, L. Mhamdi, D. McLernon, S. A. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, IEEE, 2016, pp. 258-263.

[29] A. Abubakar and B. Pranggono, "Machine learning based intrusion detection system for software defined networks," in *2017 Seventh International Conference on Emerging Security Technologies (EST)*. IEEE, 2017, pp. 138-143.

[30] S. M. Mousavi and M. St-Hilaire, "Early detection of ddos attacks against sdn controllers," in *2015 International Conference on Computing, Networking and Communications (ICNC)*. IEEE,2015, pp. 77-81.

[31] M. Z. A. Azizz and K. Okamura, "Leveraging sdn for detection and mitigation smtp flood attack through deep learning analysis techniques," *International Journal of Computer Science and Network Security*, vol. 17, no. 10, pp. 166-172, 2017.

[32] D. Santos, J. Wickboldt, L. Granville, and A. Schaeffer-Filho, "Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2016, pp. 27-35.

[33] J. Ye, X. Cheng, J. Zhu, L. Feng, and L. Song, "A ddos attack detection method based on svm in software defined network," *Security and Communication Networks*, vol. 2018, 2018.

[34] S. Lee, J. Kim, S. Shin, P. Porras and V. Yegneswaran, "Athena: A framework for scalable anomaly detection in software-defined networks," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2017, pp. 249-260.

[35] C. C. Chen, Y. R. Chen, W. C. Lu, S. Tsai, and M. Yang, "Detecting amplification attacks with software defined networking," in *2017 IEEE Conference on Dependable and Secure Computing*. IEEE, 2017, pp. 195-201.

[36] A. Alshamrani, A. Chowdhary, S. Pisharody, D. Lu, and D. Huang, "A defense system for defeating ddos attacks in sdn based networks," in *Proceedings of the 15th ACM International Symposium on Mobility Management and Wireless Access*. ACM, 2017, pp. 83-92.

[37] S. Wang, Q. Sun, H. Zou, and F. Yang, "Detecting syn flooding attacks based on traffic prediction," *Security and Communication Networks*, vol. 5, no. 10, pp. 1131-1140, 2012.

[38] C. Po-Wen, C. Kuo . Guo, and C. Lei, "How to detect a compromised sdn switch," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2015, pp. 1-6.

[39] H. Zhou, C. Wu, C. Yang, P. Wang, Q. Yang, Z. Lu, and Q. Cheng, "Sdn-rdcd: a real-time and reliable method for detecting compromised sdn devices," *IEEE/ACM Transactions on Networking (TON),* vol. 26, no. 5, pp. 2048-2061, 2018.

[40] D. Hu, P. Hong and Y. Chen, "Fadm: Ddos flooding attack detection and mitigation system in software-defined networking," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1-7.

[41] S. Seungwon, V. Yegneswaran, P. Porras, and G. 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*, pp. 413-424. ACM, 2013.

[42] H. Wang, L. Xu, and G. Gu, “Floodguard: A dos attack prevention extension in software-defined networks,” in *Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2015, pp. 239–250.

[43] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui, “Flooddefender: Protecting data and control plane resources under sdn-aimed dos attacks,” in *Proceedings of the IEEE International Conference on Computer Communications(INFOCOM).* IEEE, 2017, pp. 1-9.

[44] M. Menghao, G. Li, L. Xu, J. Bi, G. Gu, and J. Bai, "Control plane reflection attacks in sdns: new attacks and countermeasures," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 161-183.

[45] Q. Yan, and W. Huang, "A ddos detection and mitigation system framework based on spark and sdn," in *International Conference on Smart Computing and Communication*. Springer, 2016, pp. 350-358.

[46] L. Jing, Y. Lai, and S. Zhang, "Fl-guard: A detection and defense system for ddos attack in sdn," in *Proceedings of the 2017 international conference on cryptography, security and privacy*. ACM, 2017, pp. 107-111.

[47] Y.Cui, L. Yan, S. Li, H. Xing, W. Pan, J. Zhu, and X. Zheng, "Sd-anti-ddos: Fast and efficient ddos defense in software-defined networks," *Journal of Network and Computer Applications,* vol. 68, pp. 65-79, 2016.

[48] R. Durner, C. Lorenz, M. Wiedemann, and W. Kellerer, "Detecting and mitigating denial of service attacks against the data plane in software defined networks," in *2017 IEEE Conference on Network Softwarization (NetSoft)*. IEEE,2017, pp. 1-6.

[49] M. Reza, R. Javidan, and M. Conti, "Slicots: An sdn-based lightweight countermeasure for tcp syn flooding attacks," *IEEE Transactions on Network and Service Management,* vol. 14, no. 2, pp. 487-497, 2017.

[50] F. Ficherta, L. Galluccio, S. C. Grancagnolo, G. Morabito, and S. Palazzo, "Operetta: An openflow-based remedy to mitigate tcp syn flood attacks against web servers," *Computer Networks,* vol. 92, pp. 89-100, 2015.

[51] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," *in 2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 305-316.

[52] R. Kokila, S. T. Selvi, and K. Govindarajan, "Ddos detection and analysis in sdn-based environment using support vector machine classifier," in *2014 Sixth*

International Conference on Advanced Computing (ICoAC). IEEE, 2014, pp. 205-210.

[53] https://scikit-learn.org/stable/modules/neighbors.html

[54] https://scikit-learn.org/stable/modules/naive_bayes.html

[55] https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html

[56] https://scikit-learn.org/stable/auto_examples/ensemble/plot_isolation_forest.html

[57] https://blog.keras.io/building-autoencoders-in-keras.html

[58] https://scikit-neuralnetwork.readthedocs.io/en/latest/module_ae.html

[59] http://mininet.org/

[60] http://www.grid.unina.it/software/ITG/

[61] https://nmap.org/nping/

[62] https://trex-tgn.cisco.com/

[63] https://ostinato.org/

[64] https://www.wireshark.org/

[65] https://tools.kali.org/information-gathering/hping3

[66] https://resources.infosecinstitute.com/loic-dos-attacking-tool/

[67] https://resources.infosecinstitute.com/loic-dos-attacking-tool/

[68] B. L. Sturm. "Classification accuracy is not enough," *Journal of Intelligent Information Systems,* vol. 41, no. 3, pp. 371-406, 2013.

[69] Z. Li, W. Xing, and D. Xu, “Detecting saturation attacks in software defined networks,” in *Proceedings of the IEEE International Conference on Intelligence and Security Informatics(ISI).* IEEE, 2018, pp. 1–6.

[70] http://bearcave.com/misl/misl_tech/wavelets/compression/shannon.html

[71] F. Jérôme, and O. Festor. "Anomaly traceback using software defined networking," in *2014 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, 2014, pp. 203-208.

[72] A. A. Amaral, L. de Souza Mendes, B. B. Zarpelão, and M. L. P. Junior, “Deep ip flow inspection to detect beyond network anomalies,”Computer Communications, vol. 98, pp. 80–96, 2017.

[73] H. Wang, G. Yang, P. Chinprutthiwong, L. Xu, Y.Zhang, and G. Gu, “Towards fine-grained network security forensics and diagnosis in the sdn era”, in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 3-16.

[74] https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview.

[75] J. Collings and J. Liu, "An openflow-based prototype of sdn-oriented stateful hardware firewalls," *in 2014 IEEE 22nd International Conference on Network Protocols,* pp. 978-1.

[76] https://sflow-rt.com/

[77] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, V. Yegneswaran, P. Porras, and G. Gu. "Flow wars: systemizing the attack surface and defenses in software-defined networks," *IEEE/ACM Transactions on Networking (TON),* vol. 25, no. 6, pp. 3514-3530, 2017.

[78] D. Kotani and Y. Okabe, “A packet-in message filtering mechanism for protection of control plane in openflow networks,” in *Proceedings of the 10th ACM/IEEE symposium on Architectures for networking and communications systems*. ACM, 2014, pp. 29–40.

[79] L. F. Carvalho, G. Fernandes, J. J. Rodrigues, L. S. Mendes, and M. Proença, "A novel anomaly detection system to assist network management in sdn environment," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1-6.

[80] https://kiminewt.github.io/pyshark/

[81] https://linux.die.net/man/1/tshark

[82] https://github.com/netdata/netdata

[83] https://iperf.fr/