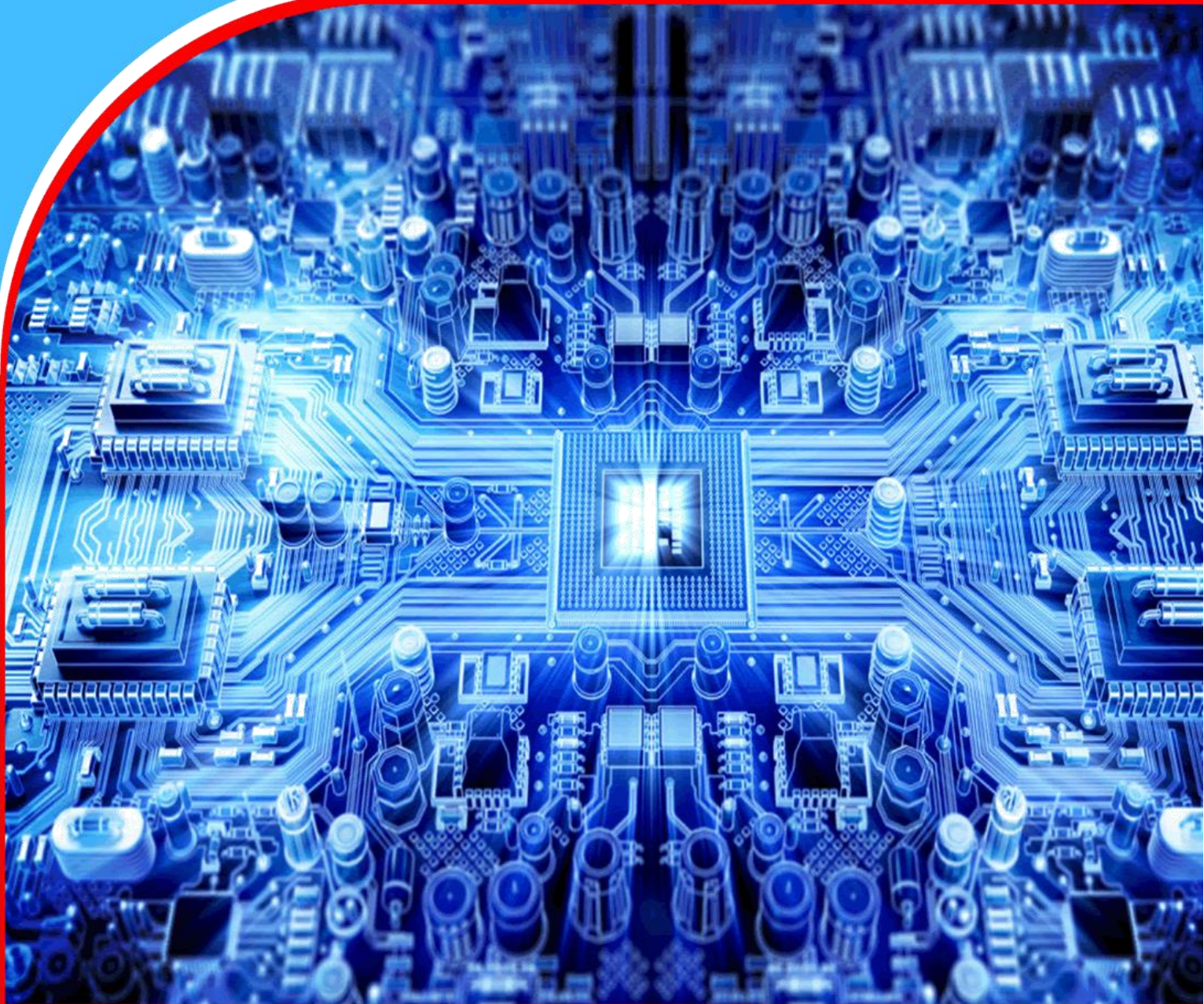


American Journal of Computing and Engineering (AJCE)



Enhanced Attacks Detection and Mitigation in Software Defined Networks

*Suh Charles Forbacha, Maah Kelvin Kinteh & Eng. Mohamadou
Hamza*



Enhanced Attacks Detection and Mitigation in Software Defined Networks

 Suh Charles Forbacha^{1*}, Maah Kelvin Kinteh² & Eng. Mohamadou Hamza³

^{1*}College of Technology, The University of Bamenda, Bambili, Cameroon

Author's Email: s.forbacha@gmail.com

²National Higher Polytechnic Institute, The University of Bamenda, Bambili, Cameroon

³Zango Enterprises, Bamenda, Cameroon



Article history

Submitted 16.04.2024 Revised Version Received 20.05.2024 Accepted 26.06.2024

Abstract

Purpose: The main aim of this research project was to develop a security simulation and mitigation mechanism for Software Defined Networking (SDN) deploying machine learning algorithms.

Materials and Methods: Applied research method was used whereby attacks were initially detected and classified using machine learning algorithms on the CiCDDoS2019 dataset; next a SDN virtual network was created through simulation in Mininet plus captured network data from the environment and finally applied machine learning algorithms to detect and mitigate the attacks in case of an attack occurrence.

Findings: Results showed higher rates of attack detection and lower false positive rates. Hence our system could be used in real life environments for attack detection and mitigation. However, the conditions and networks traffic would be different per the network configurations and tasks performed in the network environment

Implications to Theory, Practice and Policy:

Based on the findings and knowledge acquired, some key recommendations for successful implementation of an Enhanced attack and detection scheme in SDN include: Use deep learning and ensemble learning as the system will have an awareness of its state and hence have better accuracy and less false alarm rates, conducting thorough feature analysis and selection based on statistical techniques, correlation analysis, and domain knowledge, experimenting with multiple algorithms like deep neural networks, ensemble learning algorithms, optimizing the system to minimize computational overhead and ensure real-time processing, performing the study on a real world sdn environment to ensure proper knowledge of the data flow patterns in real world environments and use multiple datasets in the implementation of the system.

Keywords: SDNs, Network Management, Intrusion Detection, Prevention System, Cyber Security, Security Attacks, Attack Mitigation, Machine Learning, Real Time

1.0 INTRODUCTION

As computer networks are increasingly becoming faster and being more efficient, there has been an increasing demand for faster and more efficient networks in the past decade with many high-tech companies investing and using new network types; Software defined networking. Software-defined networking (SDN) is an approach to networking that abstracts the network control plane from underlying physical infrastructure, allowing network administrators to centrally manage and configure network resources through software (Farhady et al., 2015).

In a traditional network, network devices such as routers and switches handle both the data plane (forwarding and routing of network traffic) and the control plane (management of network configuration and policies) (Chen et al., 1993). In an SDN, control plane is separated from data plane and managed by a centralized software controller. The control plane refers to the network architecture component that defines the traffic routing and network topology while the data plane is the network architecture layer that physically handles the traffic based on the configurations rendered from the control plane. Current legacy networks have evolved into challenging monsters that are difficult to manage and lack ability to scale to today's needs of mega data centers. Software defined networking will facilitate their decoupling by separating the data, control and management planes. SDN has enhanced the programmability of networking switches by providing application programming interfaces (Jammal al., 2014).

Software-defined networking has led to significant cost reduction for consumers because devices like load balancers and firewalls implemented in current legacy networks which cost thousands of dollars are implemented as software at a fraction of the cost. SDN provides the ability to apply network virtualization based on for example on layer two or layer three features. Virtualization enables the sharing of the same network resources which helps in overall reduction of costs in large networks (Casado et al., 2007).

Software Defined Networks (SDNs) have become increasingly popular due to their flexibility and ease of management. However, the centralized control of SDNs also poses security risks, such as malicious attacks on the control plane and data plane, causing traffic slowdowns, data theft, and other security breaches (Hande et al., 2019). The market size for Software defined network is expected to reach a value of \$100 billion by the year 2025 (Cameron Magazine, 2021), with this huge market cap and investment in this industrial it is important to have the best security measures in place for such networks.

Distributed Denial of Service (DDoS) is a set of frequent cyber-attacks used against public servers. Because DDoS attacks can be launched remotely and reflected by legitimated users on networks, it is hard for victims to detect and prevent. Kumar (2020) stated that, keeping computer network safe is a task for organizations. Complex DDoS attacks are the primary vectors for causing damage to the networks, data and, availability of services. With the development of Big Data, Internet of things (IoT) and social networking applications, the network traffic has increased many folds and so are the network threats.

Traditional Intrusion Detection Systems and Intrusion Prevention systems are not the most appropriate security mechanisms to identity advanced attacks and per say Zero-Day attacks. Therefore, there is a need to devise new strategies, which can go up against these threats. The other real obstacle in utilizing existing strategies is 'human intervention', which is required at present for threat recognition. With the establishment of a large campus-wide area network, internet facilities, Wi-Fi attack-surfaces have multiplied (Farina et al., 2015) One of the well-known applications that use the rule-based approach is Snort, a packet filter tool developed by Roesch. This tool can filter packets by analyzing many characteristics of each packet that flows on a server or host. For example, it can filter by source IP, destination IP, port number, protocols, packet size and content. Many

researchers have enhanced this tool or integrated this tool into their design. Because Snort filters the packets using rules, the false alarm rate is low (Roesh, 1999). Snort is therefore a foremost Open Source Intrusion Prevention System (IPS) in the world. Snort IPS utilises a series of rules that help define malicious network activity and deploys those rules to find packets that match against them and generates alerts for users.

Machine learning algorithms have also shown great potential in cybersecurity applications, including attack detection and threat mitigation. Machine learning techniques have equally been deployed to examine huge datasets to find patterns and forecast the probability of an attack and create a mitigation model for the attack. (Junhong, 2020). The Security industry is experiencing tremendous attacks and needs to respond with robust mechanisms that protected them from such cyber-attacks

Problem Statement

Security is a pertinent issue in networks because if one gets in the position of control of computer networks, it means they can access nearly all other resources at a company's disposal (Garg et al., 2019). The main problem addressed in this research is the lack of effective security mechanisms in Software Defined Networks that can detect and mitigate security threats in real-time. Existing security solutions are often static and unable to adapt to dynamic changes in the network environment.

The security of software-defined networks (SDNs) can be compromised by various types of attacks, such as distributed denial-of-service (DDoS) attacks, intrusion attempts, and malware infections. Traditional security mechanisms such as firewalls and intrusion detection systems may not be effective in detecting and mitigating these attacks, especially in large-scale networks (Sahay et al., 2017). This research was aimed at providing a security and mitigation framework that uses machine learning to provide a dynamic and adaptable system for the detection of Distributed Denial of Service attack in Software Define Networks. Despite the increasing benefits of Software Defined Networking and it continues application and implementations in various big enterprises, it possesses some security risks. This research work aims to addressing the security challenges faced by Software Defined networking environment and providing solutions by developing machine learning models for attack detection and attack mitigation with higher accuracy and less false positive rates.

- i. To reduce the total damage an attack on a network can cause
- ii. To detect attacks in real time before an attacker can have total control of a system
- iii. To use new means to train a controller to detect an attack from normal bulk traffic and attacks

Objectives and Research Questions

The main objective of this research was to develop a security simulation and mitigation mechanism for Software Defined Networking deploying machine learning algorithms. In order to accomplish our main objective, the following specific objectives were used.

- i. To identify the security threats in Software Defined Networking and their impact on the network.
- ii. To develop a Network to mimics the attacks and have them mitigated
- iii. To develop a simulation framework to generate realistic network traffic and security threats
- iv. To train machine learning models to detect security threats in the simulated network traffic.
- v. To develop a mitigation framework to respond to the detected security threats and potentially stop the attack.

In this research paper, the researchers explored vulnerabilities in software-defined networks focusing on a single point of failure - the controller since it is the brain of the SDN.

Main Research Question

How can the researcher develop a mechanism that can simulate various attacks, detect such attacks and have a mitigation means in real time for Software Define Networks using machine learning algorithms?

Specific Research Questions

- i. How can an attack be simulated in software defined networking environment?
- ii. How can network traffic be developed to mimic both normal and malicious traffic?
- iii. How can machine learning techniques accurately detect and classify different types of attacks in Software Defined Networks, such as distributed denial-of-service (DDoS) attacks, and control plane attacks in real time?
- iv. What mechanism can be developed to mitigate such attacks in real time and potentially stop the attack?

Software Defined Networking

Software Defined Networking (SDN) is networking architecture approach which enables the control and management of network using software applications. Through SDN networking, the behavior of entire network and its devices are programmed in a centrally controlled manner through software applications using open Application Programming Interfaces (API). SDN improves performance through network virtualization. In SDN software-controlled applications or Application Programming Interfaces work as basis of complete network management that may be directing traffic on network or to communicate with underlying hardware infrastructure. So, in simple terms, we can say SDN can create virtual network or it can control traditional network with the help of software (Yang et al., 2020).

Software-Defined Networking is often described as the ‘End of Networking’ these days William et al. (2018). This rapidly growing technology has been a huge success owing to the separation of control plane from the data plane of the networking devices. The control plane allows management of the entire network from a global point of view, thereby eliminating the need of a separate controller in every networking device. The data plane, consisting of the switching elements allows forwarding of network traffic based on rules programmed by the applications running on top of the controller. This significant advancement in the field of networking has helped accelerate service delivery and provide more agility in provisioning both virtual and physical network devices from a central location. The centralized controller was responsible for enforcing global policy whereas the Switches simply forwarded packets based on rules in a flow table. This allowed data and control plane to be separated thereby allowing more programmability. The networking world could sense the arrival of SDN just after the Ethane architecture was proposed (Casado et al., 2007)

Figure 1 below illustrates the architecture of Software-Defined Networking

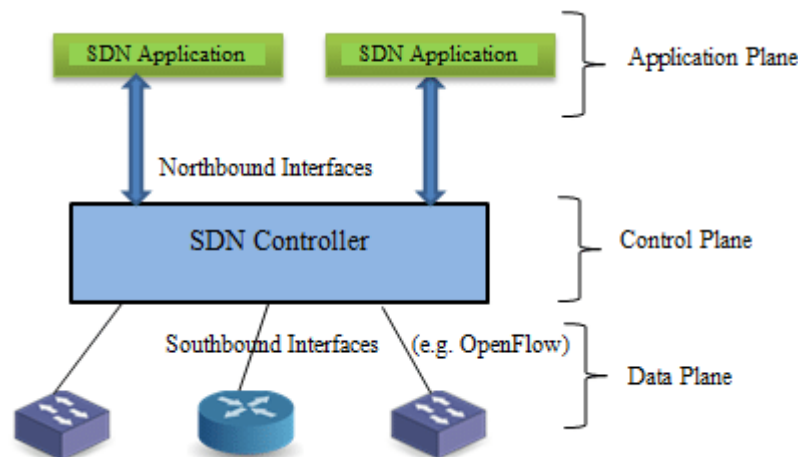


Figure 1: SDN Architecture (Sangeeta, 2018)

Control plane takes up the role of global management of the network. It maintains global network topology and instructs the data plane devices by providing them with flow instructions. The data plane networking devices forward the traffic based on the flow rules installed in them. SDN uses Northbound API to communicate with the applications and business logic present in the application layer to help the network executives for traffic management service deployment. The controller uses Southbound API (e.g. OpenFlow, NetConf) to establish a secure communication channel with one or more SDN compatible switches for monitoring and controlling the traffic forwarding.

(Neuphane et al., 2018) noted that SDN was born out of the need to break the vertical integration of the network equipment. Its idea was to separate the control from the data plane and OpenFlow (OF) protocol, proposed in 2008, which leveraged its development. It also allows defining network functions (e.g., routing, firewall, load balancing, bandwidth optimization) as software applications that can run on top of the control plane. The architecture has three parts: data plane (composed of switches), the control plane (composed of one or more controllers), and application plane (composed of one or more network applications). This new paradigm represents a solution to several problems of traditional networks, such as manageability, configuration, scalability, and security. Under this perspective, a clear advantage for security with SDN is the ability to gather traffic information without additional elements. This is due to the centralized role of the controller, which communicates with the switches in the data plane.

There are two trends currently in SDN: those focusing on the dynamic virtual machine migration and use of hypervisors as well as techniques such as encapsulation and tunneling and those that are striving to accomplish software control of the network by using the OpenFlow protocol to manipulate the flow tables in switches (Metzler, 2014). VMware and Open Networking Foundation are for centralization of network control and don't see the role of hardware for some network function in data centers while on the other hand is Cisco which supports both trends.

In 2017, during an Open networking summit, Google presented its espresso SDN pillar which is extending its SDN to the edge of the public internet and as a result making its cloud 25% faster, more available, and cost-effective (Hardesty, 2017). Hardware SDN device vendors include Cisco, HP, Jupiter, Big Switch, and Netgear. Open Networking Foundation (ONF), the organization promoting the adoption and implementation of SDN as an open standard developed the OpenFlow standard. OpenFlow management and configuration protocol standard a first of its kind is a vendor-neutral interface between the control layer and data layer of the SDN architecture (open networking foundation).

Software-defined networking architecture is dynamic, manageable, cost-effective, and adaptable. These qualities make it suitable for today's bandwidth-hogging data centers that are in need of a

flexible solution that can put resources where they are needed. An SDN architecture is considered to be:

- i. Programmable- this is facilitated by the decoupling of control and the forwarding planes.
- ii. Agile -This is brought about by the abstraction of the control plane from the forwarding plane which gives network engineers freedom to make network-wide changes in traffic flow to meet their current needs.
- iii. Central management - SDN has the concept of a controller that has a network-wide view that appears to all applications and policy engines as a single switch. This visibility makes network management easier.
- iv. Programmatically configured - SDN gives freedom to network engineers to configure, manage, secure, and optimize network resources very quickly using dynamic custom SDN programs that are vendor independent and can be written by themselves.

In SDN, a northbound interface is an application programming interface (API) or protocol that permits a lower-level network component to communicate with a higher-level or more central component, while -- conversely -- a southbound interface allows a higher-level component to send commands to lower-level network components. Northbound and southbound interfaces are most associated with software-defined networking (SDN), but can also be used in any system that uses a hub-and-spoke or controller-and-nodes architecture. North and south in this context can be construed of as on a map. The north is on the top and south on the bottom of the diagram. The higher-level elements control the lower-level ones. Examples of southbound APIs include OpenFlow, CISCO, OpFlex while those of northbound include.

Controllers

Controllers are the brain of a software-defined network that takes the control plane out of network hardware and runs it as software. The controller facilitates easier integration, administration of applications and automated management of networks. It has a global view of all forwarding devices in the forwarding plane. It also has a way to communicate forwarding instructions to the forwarding devices via the southbound API. It provides abstractions of the network to all the network applications which makes developing of network applications easier as developers need only to know how to interface with the controller. The controller is vital to SDN because it performs those control functions that were previously done by switches. (Casado et al., 2007). If packets do not match against the flow tables, it is the controller that directs switches on the destination port or address of the packets through the packet-out message. This is possible because the controller has an entire view of the network. Some important controllers include:

Open Daylight

OpenDaylight is a community-led and industry-supported open source SDN project initiated by the Linux Foundation with the sole aim to advance software-defined networking adoption and make a strong case for network function virtualization. It aims to deliver readily deployable controller without any need for other components. It supports add-ons that can add value to its uses. It is java based, and some of its founding members are Big Switch, Cisco, Brocade, Ericson, HP, and IBM. It deploys open standards, and as a result of working with open networking foundation, it readily supports OpenFlow though its open to any future open protocols other than OpenFlow. It provides a northbound API/Restful API which can be used to effortlessly develop applications network applications. (Opendaylight Project).



OpenDaylight Architecture - Operational View

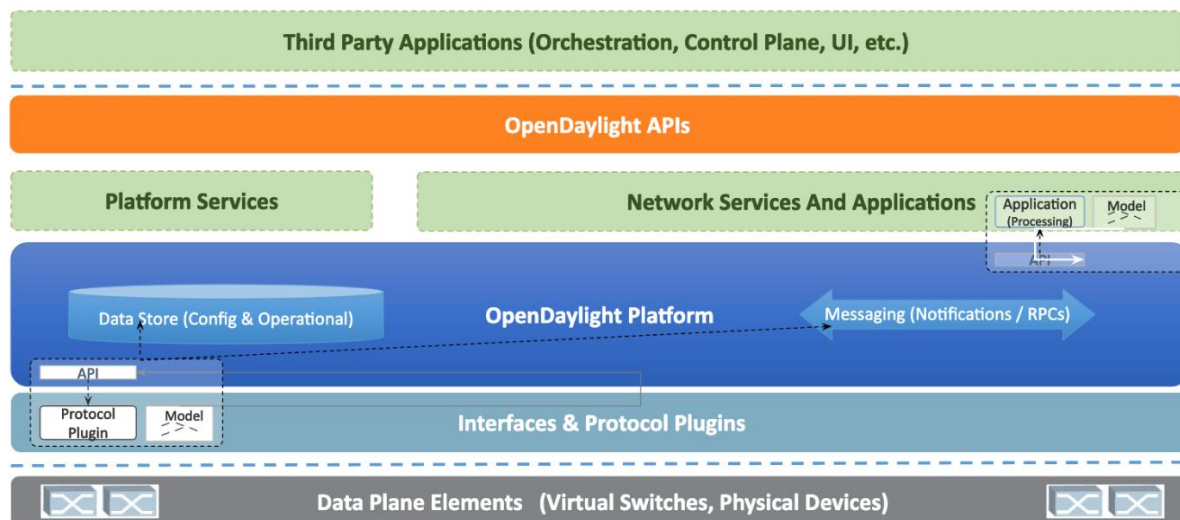


Figure 2: Open Daylight Architecture (Gonzalez, 2017)

Floodlight Controller

It is a Java based open source enterprise controller developed by Big Switch. It is easy to use, has a large community of users thus highly likely to get help. It's easy to use and easy to set it up because of the few dependencies. It's multithreaded therefore has high performance, it supports OpenStack making it easily deployable in cloud computing scenario. It also supports both Open Flow and non-Open Flow switches hence highly applicable in every network scenario (Project Floodlight)

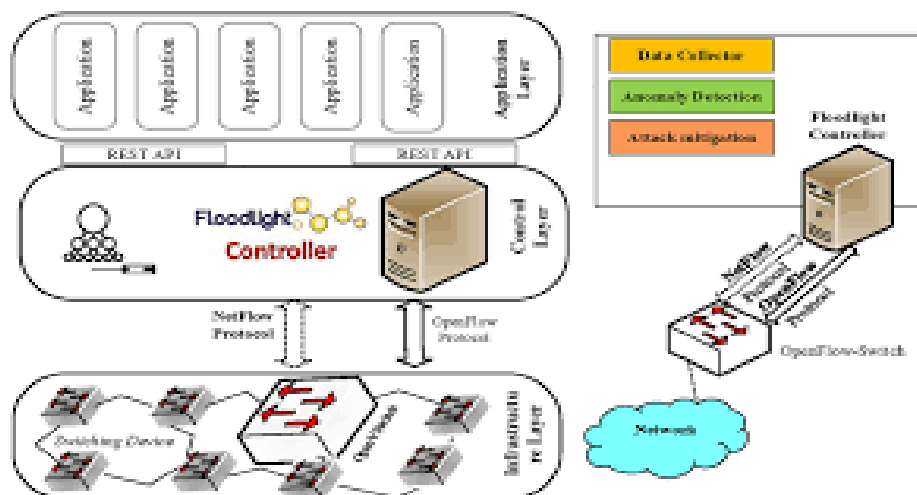


Figure 3: Floodlight Architecture (Jafarian et al., 2020)

Pox Controller

Pox is a python based open source software-defined networking controller which is very popular for rapid prototyping. It comes with components such as a hub, a layer three switch component, topology discovery and even a spanning tree component which all contribute towards rapid prototyping.

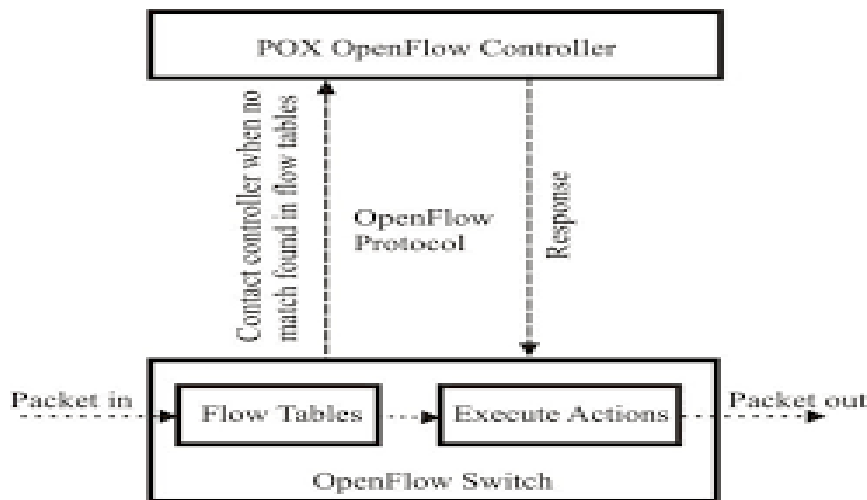


Figure 4: Pox Controller Architecture (Kaur et al., 2014)

RYU Controller

Ryu (means flow in Japanese) is a component-based software defined networking framework. Ryu provides software components with well-defined API that make it easy for developers to create new network management and control applications. Ryu supports various protocols for managing network devices, such as Openflow, Netconf, OF-config.

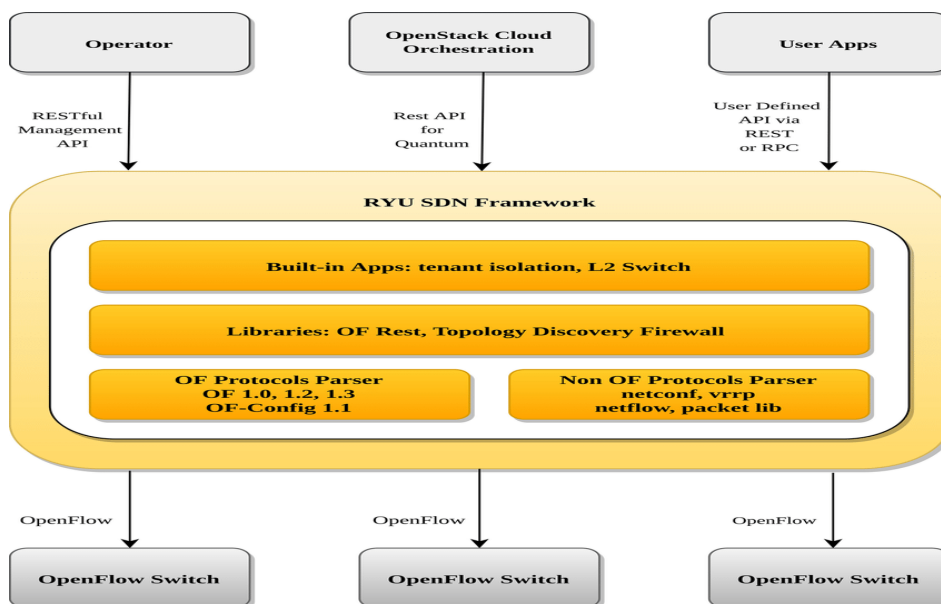


Figure 5: RYU Controller Architecture Author (Islam et al., 2020)

Secure Channel

The secure channel facilitates the interaction between the controller and the OpenFlow switch. It enables the controller to be able to configure and manage the switch by receiving events from the switch and facilitating sending of packets out of the switch. It uses TLS protocol for secure communication. The protocol is the lifeline of an SDN network because if it the connection fails, the packets which don't match with the switch flow table will not be sent to the controller. If the switch

cannot establish a link to the backup controllers, it will fall into the fail secure mode, and all packets that do not match against flow entries are dropped instantly.

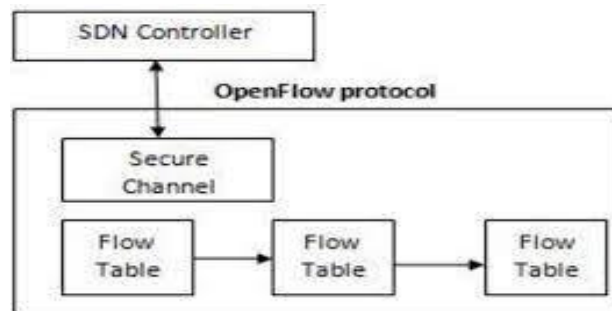


Figure 6: Secure Channel Architecture (Islam et al., 2020)

Southbound Interface

The controller communicates with network devices via the southbound API. The API is used to transmit information such as packet handling instructions, notifications on status changes, e.g., if some devices or links are up or down and statistics information such as flow counters or aggregate statistics. Open Flow is the most common protocol used in the SDN.

Open Flow

This is the standardized communications protocol that facilitates interaction between the control plane and the forwarding plane. It is the southbound API for software defined networks. It enables network programmers to modify the behavior of switches and routers through writing scripts that run in the controller. Open Flow protocol allows direct access and manipulation of forwarding plane devices such as switches and routers. Other than Open Flow other important SDN protocols include:

- i. Border gateway protocol for hybrid SDN.
- ii. NETCONF which is mandatory for configuring Open flow enabled devices.
- iii. MPLS-TP a transport profile for multiprotocol label switching used as a network layer technology in transport networks.
- iv. Open v-Switch Database Management Protocol (OVSDB) - an Open Flow configuration protocol for managing open v-Switch implementations in SDN.
- v. Extensible Messaging and Presence Protocol (XMPP) which is used for messaging and online presence detection all in real time

Open Flow Messages

They facilitate communication between the controller and switches. Open flow messages also lead to particular events being invoked. Some of the messages include:

- i. Features request: It's sent by the controller to the switch. It is composed of just the Open Flow header.
- ii. Features reply: The switch replies to the ofpt_features_request by the controller using this message.
- iii. Packet out: The controller sends this message to the switch instructing it to send a packet or enqueue it or even discard it. Its attributes include buffer_id, in_port, actions and the data (bytes).
- iv. Flow modification message: It's a message with instructions for modifying the flow table. The crucial fields in flow modification message are hard_timeout, and idle_timeout in that

they determine how fast flows expire. These fields can be used to write flows or delete flows that are suspect. It's sent by the controller to the switch.

- v. Port modification message It's used to modify the behavior of physical ports.
- vi. Statistics messages- Flow statistics messages include:
- vii. Individual Flow Statistics-Individual flow stats.
- viii. Aggregate Flow Statistics- Contains multiple flows statistics.
- ix. Table Statistics- Contains stable information.
- x. Port Statistics- Contains physical statistics.
- xi. Packet-In Message The packet-in message is sent from the switch to the controller. It's active when packets arrive in the datapath or switch and don't match all fields thus sent to the controller to determine appropriate actions to be performed on the packets. The actions include to forward the packet to a particular port, to drop the packet or modify the packet headers.
- xii. Flow Removed Message Flow Removed message is used by the datapath/switch to inform the controller that a flow has been removed.

Northbound API

SDN applications include load balancers, firewalls, security applications, or applications like OpenStack that operate in cloud services or in the networks. The northbound API provides network control information to these applications that have very high instance abstractions of the network. The value of SDN is enhanced by the availability of the northbound API on top of which the SDN applications will be developed otherwise the control plane, and forwarding plane are doing the same things. SDN has the upper hand because, with the centralized view of the network by the controller, all control information is in one place and is made available via an API. (sdxcentral.com, 2023)

Security Vulnerabilities in Software Defined Networking (SDN)

SDN has numerous applicability in current networks and future networks such as 5G networks. SDN is being used for cloud networking and gateway control. However, SDN security faces a lot of threats from malicious applications, attacks from the forwarding plane as well as attacks from the control network through the northbound APIs in a cloud computing environment (Schneider, 2015).

The centralized view of the network is also a single point of attack and failure. The southbound API is also prone to attacks that can compromise its availability as well as performance (Lim, 2015). Despite the advantage of programmability and the ability to manage packet forwarding and policy application, SDN security is considered a crucial issue as noted by Lim (2015). Due to SDN infancy and inability to enforce security on a physical topology, an SDN attacker can identify targets, modify content, conceal from intrusion detection systems, attack servers, and monitor traffic leaving SDN operators exposed. Lim (2015) also points out that other factors such as the use of Transport Layer Security (TLS) for encryption and authentication can leave an SDN network susceptible. Lim further noted that the possibility of controller flooding and ability to impersonate an OpenFlow switch is a real security headache as is the availability of debugging ports that are not encrypted thus can be used to take control of the switch. There are several vulnerabilities in the SDN ecosystem.

Hinden (2014) noted that the biggest threat to SDN environment is compromising of the controller because if the controller is compromised, then the entire network is compromised. Effects of SDN controller being compromised include controller subverting new flows, ability to launch a man in the middle attacks, ability to modify content on the network, monitoring of traffic and sending traffic to compromised nodes inadvertently (Hinden, 2014).

In addition to major vulnerability in Software-Defined networking, are the following vulnerabilities:

Configuration vulnerabilities: These vulnerabilities are related to the incorrect configuration of network devices, such as firewalls, routers, switches and servers. For instance, a misconfigured firewall may allow an attacker to access internal resources.

Software vulnerabilities: These are security vulnerabilities in software that can be exploited by attackers. Examples include security loop holes in operating systems, web browsers, productivity applications and web applications. Attackers takes advantages of such instances to compromise a computer network.

Protocol vulnerabilities: These vulnerabilities are related to security loop holes in the communication protocols used by computer systems.

Physical vulnerabilities: These vulnerabilities are related to physical equipment used in computer networks, such as network cables, switches, routers and servers. Examples include security flaws in security locks, encryption keys and surge protectors.

Human vulnerabilities: These vulnerabilities are related to human behaviour that can compromise the security of computer networks. Examples include reusing passwords, opening phishing emails, downloading malware and sharing sensitive information.

Network Attacks

An attack is defined as a malicious interaction designed to violate one or more security properties. It is an external fault created with the intention of causing harm, including attacks launched by automatic tools. These include denial of service attacks (DoS), password attacks, man-in-the-middle attack, computer virus and social engineering which are but a few to be stated.

Distributed Denial of Service (DDoS)

A distributed denial-of-service (DDoS) attack is a malicious attempt to disrupt the normal traffic of a targeted server, service, or network by overwhelming the target or its surrounding infrastructure with a flood of internet traffic. DDoS attacks achieve effectiveness by utilizing multiple compromised computer systems as sources of attack traffic. DDoS attacks are considered one of the most common threats nowadays, posing critical danger to network services due to their ease in carrying out and compromising the availability of services in seconds. As analyzed by recent reports, DDoS attacks have grown rapidly over the last several years, which has caused significant financial losses to the business. The difficulty in locating the assailant provided a considerable boost to the attack's efficacy. For instance, the Mirai botnet attack in 2016 affected network availability over the globe, which brought down a large portion of the Internet (Kolias et al., 2017).

In the year 2018, GitHub servers were targeted by the largest-ever DDoS attack. In another case, the attack using application layer protocol produced 129 million requests per second and achieved a total traffic level of 1.35 Tbps sent in the attack, which followed the severe attack. (The Guardian, 2023) A common technique used to perform DDoS attacks is IP spoofing. Spoofed (fake) IP is used for the requests, and the destination host machine tries to send a response to every single request. This results in response being sent to almost every address in the IP pool. With the proliferation DDoS attacks in a number of countries, the need for deploying a protective mechanism against the same has become increasingly important.

Intrusion Detection System (IDS)

Intrusion detection is the process of monitoring network traffic or computer events to detect malicious or unauthorized activities. Any device or software application whose main purpose is to perform intrusion detection is an intrusion detection system. They function by analysing traffic and reporting

logs and cannot take any decision to block the attack (Mell, 2007). Some examples include: network-based IDS, host-based IDS, application-based IDS, signature-based and anomaly-based IDS.

Intrusion Prevention System (IPS)

It is the act of extending the roles of an intrusion detection system to include the ability to block the malicious unauthorized activities detected (Cisco Systems). Intrusion prevention involves deep packet inspection which can alleviate different network attacks, worms, and viruses. Some of the IPS systems have advanced features such as real-time sandboxing and mitigation solutions, global threat intelligence and intelligent security automation. IPS sits behind the firewall to complement it in filtering out dangerous content (Simkin, 2020).

Actions by Intrusion prevention systems include:

- i. Sending alarm to the administrator.
- ii. Dropping Malicious packets.
- iii. Blocking traffic from the source address.
- iv. Resetting connection

Machine Learning

Machine learning (ML) is a branch of artificial intelligence that enables computers to learn from data without being explicitly programmed. In network security, machine learning has been increasingly used for attack detection and attack mitigation. Decision Tree (DT), K-Nearest Neighbor (KNN), Artificial Neural Network (ANN), Support Vector Machine (SVM), K-Means Clustering, Fast Learning Networks, Ensemble Methods which are but a few to be stated are the most popular ML methods used for DDoS detection in SDN (Eliyan et al., 2021)

Developing a machine learning algorithm involves two specific steps: training and testing. Each model has its own training techniques. This process and the design of machine learning model is usually managed by a framework such as scikit-learn, Tensor Flow, Py-Torch, MATLAB or Weka. The framework used has a strong impact on the optimization of the algorithm or the number of available parameters. Machine learning models can perform many tasks, two of which are particularly interesting for intrusion detection: classification and regression. Classification categorizes entries into several classes, such as “normal” or “attack”, or even different families of attacks. Regression (also called “prediction”) is used to determine continuous values, including a probability that an input is an attack (Sarioguz, & Miser, 2024).

Machine learning can be classified as either supervised, unsupervised and reinforced learning.

Supervised Machine Learning

Supervised machine learning is applicable where the labels and feature of the training set are known but need to be accurately predicted in other unlabeled data. These algorithms use data that is labeled as for example ‘Anomaly or ‘Normal’ traffic as training data to make models which are used to predict class instances in the test dataset. To test the efficiency of an algorithm, the predicted outcomes are compared with real data (Le, 2016)

Unsupervised Learning

This type of machine learning involves algorithms that train on unlabelled data. The algorithm scans through data sets looking for any meaningful connection. The data that algorithms train on as well as the predictions or recommendations they output are predetermined.

Semi-Supervised Learning

This approach to machine learning involves a mix of the two preceding types. Data scientists may feed an algorithm mostly labelled training data, but the model is free to explore the data on its own and develop its own understanding of the data set.

Performance Metrics in Machine Learning

Several metrics are used to describe the performance of a machine learning classifier

- i. True Positive (TP) Attack records correctly classified as attack records.
- ii. True Negative (TN) Benign records correctly classified as benign records.
- iii. False Positive (FP) Normal records incorrectly classified as attack records.
- iv. False Negative (FN) Attack records incorrectly classified as benign records.
- v. Detection rate (or “true positive rate”, “recall”, “sensitivity”) is the proportion of attacks that are correctly detected.

$$\text{Detection Rate} = \frac{\text{True Positive}(TP)}{\text{True Positive}(TP) + \text{False negative}(FN)}$$

False positive rate (or “false alarm rate”) is the proportion of normal traffic incorrectly flagged as attack.

$$\text{False Positive Rate} = \frac{\text{False Positive}(FP)}{\text{True Negative}(TN) + \text{False Positive}(FP)}$$

Accuracy is the fraction of correctly identified results (attack and normal traffic). In multiclass classification, accuracy is equal to the Jaccard index, which is the size of the intersection divided by the size the union of the label sets.

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

Precision (also called positive predictive value) is the proportion of identified attacks that are indeed attacks.

$$\text{Precision} = \frac{\text{True Positive}(TP)}{\text{True Positive}(TP) + \text{False positive}(FP)}$$

Recall (TPR) The proportion of correctly detected positive values.

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

F1-score is the harmonic mean of precision and recall (previously called “detection rate”).

$$F1 - \text{score} = \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Confusion Matrix a $N \times N$ Matrix that helps summarize how successful the model was in predicting attacks, where N is the number of unique labels. Higher values across the primary diagonal indicate better results.

Learning Curve: A learning curve shows the effect of increasing the size of the training data on the score of the model. It can be used to infer overfitting and bias in the training data.

ROC Curve a Receiver-Operating Characteristic (ROC) Curve shows the relation between the TPR and FPR at varying classification thresholds. A steeper curve towards the y-axis represents better detection by the model.

Area Under the Curve (AUC) The Curve refers to the ROC Curve. The AUC is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (Fawcett, 2020)

Related Works

A protocol-based network intrusion detection system was designed by Patil et al. (2018) to detect DoS/DDoS attacks in networks. In this system, Incoming packets were distributed according to the protocol and queued for additional processing. Relevant features were extracted, and protocol-specific classifiers were applied on each packet to generate alerts and thus update the attack signature database. This approach focused on detecting DoS/DDoS attack types. The main features that the classifiers focused on were the types of protocols.

Modi et al. (2012) proposed a NIDS that integrated Naive Bayes classifier and Snort. In this framework, Snort signature-based detection filtered the captured packets. The captured packets were divided into two sets: intrusion packets and non-intrusion packets. The intrusion packets were logged and denied by the system. Meanwhile, the non-intrusion packets were be preprocessed and fed into the anomaly detection module. The anomaly detection module employed the Naive Bayes classifier to further classify the non-intrusion packets into normal and intrusion packets. Once the packets were classified as intrusions, they were logged and denied. Only when the packets are labelled as normal can they be allowed to go to the system. The F-1 score tested on the KDD'99 dataset varies from 91.25% to 98.01%.

In another study carried out by Kousar et al. (2021), they discussed about the different machine learning approaches which could be used with software defined network to detect and mitigate DDOS traffic in a network. The authors demonstrated the implementation of four ML algorithms namely, Multilayer perception MLP, Decision tree, Support vector machine(SVM) and random forest. These ML algorithms were simulated using mininet. The results showed that random forest algorithms achieved the best accuracy and decision tree algorithms gives the best processing time for the DDOS attack detection. However, there were few drawbacks in the implementation to classify flow table attack and bandwidth attack.

Shoeb and Chithralekha (2016) proposed a solution intended to defend the control and data planes from DDoS attacks, established a controller process priorities set according to the node trust level, which was configured based on the node behaviour during regular business hours. The node worth was calculated based on its activity. In high-demand situations, the controller is set up to ignore requests from some nodes if the sum of their requests has already reached a predetermined maximum. The controller makes a rule change to one with a shorter timeout, prompting a response from the standard nodes as well.

Kokila et al. (2014) proposed a Support vector machine (SVM) classifiers for detecting DDoS attacks. The SVM must be trained with historical data before it can reliably predict the behaviour of unobserved traffic samples. When compared to other simulated methods, the SVM had superior accuracy and fewer false positives. However, SVM is very dependent on the accuracy of the data used to train the model. Sahay et al. (2017) proposed A DDos security architecture with its primary goal to reduce the amount of destructive Internet traffic. In order to identify network flows, the customer end detection engine was the one that determines whether or not the traffic flow is malicious. The status of the connection is communicated from the controller belonging to the customer to the controller belonging to the service provider. A determination made by the ISP controller directs that the harmful flow be sent to the filter so that it can be examined in further detail. Nevertheless, the communication between controllers also needs to be secure.

In another study carried out by Myint Oo et al. (2019), they proposed an advanced support vector machine based ML algorithm. In this study Advanced SVM algorithm collects data from feature <https://doi.org/10.47672/ajce.2120>

extraction stage and classifies parameters to predict DDoS attacks in SDN and noted that their technique reduced the testing and training time for the machine learning algorithm to perform its tasks. The implementation was done on opendaylight controller and simulation environment implemented using mininet. The authors noted that Advanced SVM method had a detection accuracy of 97% with the fastest testing and training time.

A real-time machine learning approach was used by Amrish et al. (2022) whereby they collected data features from network packet headers which were in turn classified using machine learning algorithms such as decision trees, KNN, Random forest algorithms among others. They equally had a post processing module that helped in dealing with false positives and outliers. It is worth noting that most of these research projects from the related works addressed DDOS attacks in SDN but did not look at other forms of attacks. This therefore provides research gaps that provides a platform for further studies

2.0 MATERIALS AND METHODS

Research Method

In this paper, we utilised the Applied research method. This research method is aimed at solving a particular problem or providing innovative solutions to pertinent issues that an individual, group or society face. It is referred to as applied research because it involves a practical application of scientific methods in solving a particular problem. When conducting applied research, the researchers took extra care to identify a problem, develop a research hypothesis or research questions and went ahead to test these hypotheses or answer these questions via an experimental implementation. In many cases, this research approach employed empirical methods in order to solve practical problems. So this paper deployed action research which is set at providing practical solutions to problems. Figure 7 depicts the steps deployed.

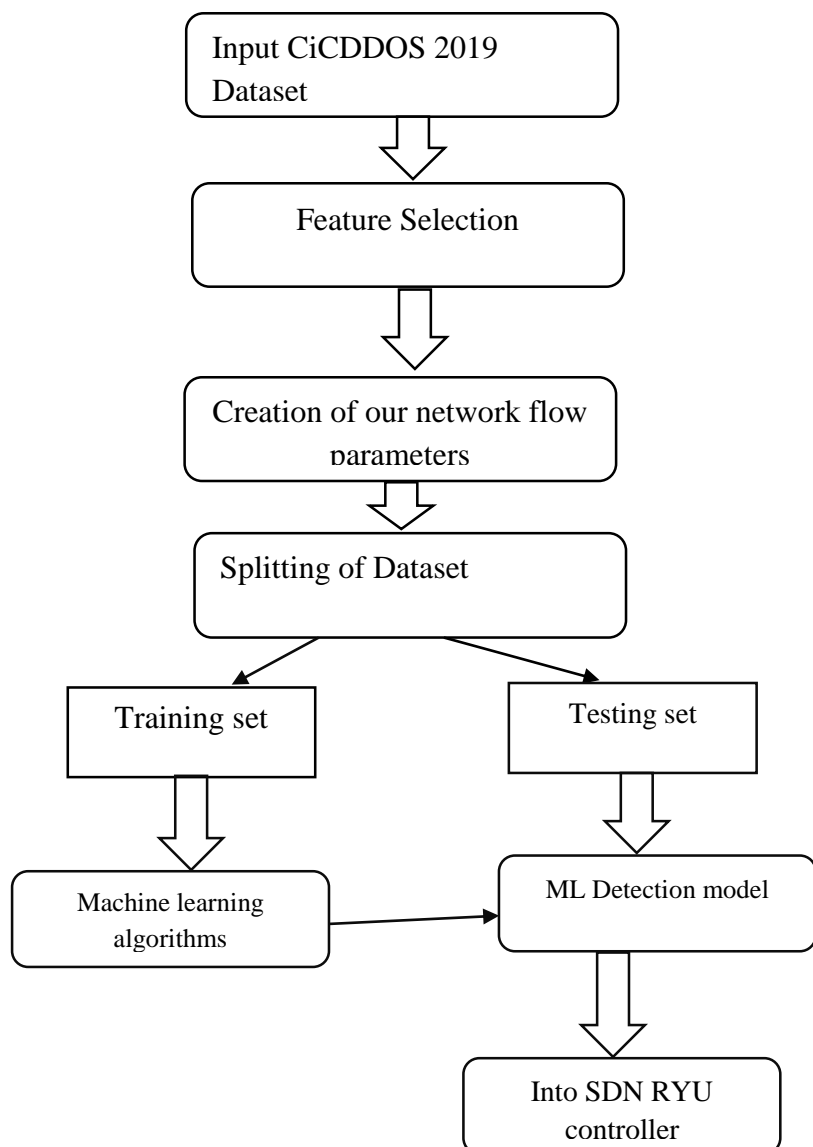


Figure 7: Methodology Flow Chart

Machine learning as a fast growing field is increasingly having a great impact in the implementation of security schemes for attack detection and mitigation. Enhancing the act of attack detection and mitigation in Software Defined Networks is of great importance to the effective functioning of companies and the day to day running of business. In this research work, we focused on the implementation of machine learning algorithms in the field of Cybersecurity, specifically in the new domain of Software defined networking. Machine learning stages involve; Framing the problem, collecting the data, Feature extraction, Model building, Model Evaluation and finally its implementation.

Data Collection; Dataset for the Project

The dataset used in this paper is CiCDDoS2019 (Sharafaldin et al., 2019). This dataset was downloaded from the Canadian institute for cybersecurity website (*Canadian Institute for Cybersecurity / UNB*). This dataset contains only DDoS attacks and benign traffic. The creators described it as a realistic cyber defense dataset. This dataset is a joint project of the Canadian Communications Security Establishment (CSE) and The Canadian Institute for Cybersecurity (CIC). CIC-DDoS 2019 is a new, high quality, synthetic dataset, providing both network traffic and log data. In order to generate the dataset, networks of targeted machines were instantiated via AWS and <https://doi.org/10.47672/ajce.2120>

automated using CIC-BenignGenerator. These machines represented five departments of a target organization, with 420 clients and 30 servers in total. Targeted machines were instrumented and then systematically attacked using an attack infrastructure of 50 machines, with log data and network traffic data captured and categorized. There is evidence of considerable effort on the part of the dataset’s creators to enhance external validity through their choice of architecture, the design of both target and attack networks, and their experimental design (Canadian Institute for Cybersecurity)

Benign

In this dataset, there are also benign data. The authors used CIC-BenignGenerator to imitate benign background traffic based on the profiles of abstract behaviour of 25 users. Benign traffic is based on HTTP, HTTPS, FTP, SSH, and email protocols. (Sharafaldin et al., 2019)

Attacks

Two genres of DDoS attacks are captured in this dataset. The first is Reflection-based DDoS, including MSSQL, SSDP, NTP, TFTP, DNS, LDAP, NetBIOS and SNMP. In this type of attack, the real attackers can hide behind the legitimated clients and utilize them in an attack. It makes the victims more challenging to differentiate the users and attackers only by the source. These attacks are based on TCP (MSSQL and SSDP), UDP (NTP and TFTP) or both (DNS, LDAP, NETBIOS and SNMP). The second is Exploitation-based attacks, including SYN flood, UDP flood and UDP-Lag. This type of attack will spoof the source IP address and sent a large number of packets to the victim server. This will cause the victim resources exhausted. A testbed architecture to execute this is shown in Figure 8

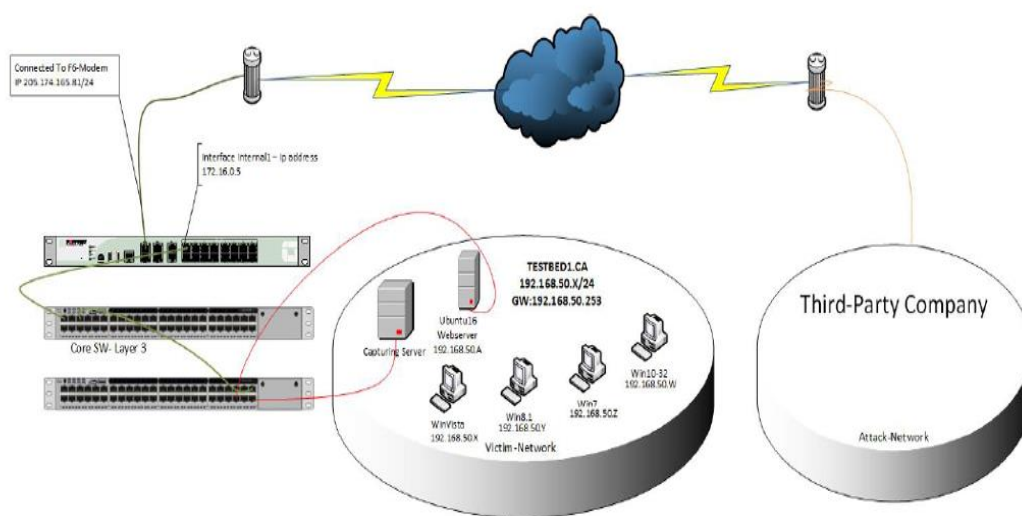


Figure 8: CiCDDoS2019 Testbed Architecture (Sharafaldin et al., 2019)

Table 1: Attack Time Zone for CiCDDoS2019 Day 1

Days	Attack	Attack time
First Day	portMap	9:43 -9:51
	NetBIOS	10:00 – 10:09
	LDAP	10:21 – 10:30
	MSSQL	10:33 -10:42
	UDP	10:53 – 11:03
	UDP-LAG	11:14 -11:24
	SYN	11:28-17:35

Table 2: Attack Time Zone for CiCDDoS2019 Day 2

SECOND DAY	NTP	10:35 – 10:45
	DNS	10:52 – 11:05
	LDAP	11:22 – 11:32
	MSSQL	11:36 – 11:45
	NetBIOS	11:50 – 12:00
	SNMP	12:12 – 12:23
	SSDP	12:27 – 12:37
	UDP	12:45 – 13:09
	UDP-Lag	13:11 – 13:15
	WebDDoS	13:18 – 13:29
	SYN	13:29 – 13: 34
	TFTP	13: 35 – 17:15

In this dataset, the authors provided two types of data for researchers. One was generated CSV files, and the other one was raw PCAP files captured from their experiment. The dataset consists of 80 features which we have discussed in the Table 3.

Table 3: Features of Dataset

Feature name	Description
Flow Duration	Time taken for a complete flow in the network
Total fwd packet	Total packets in forward direction
Total bwd packet	Total packets in backward direction
Total length of fwd packets	the total size of packets in the forward direction
Fwd packet length max	Maximum size of packets in the forward direction
Fwd packet length min	Minimum size of package in forward direction
Fwd packet length mean	the average size of packages in the forward direction
Fwd packet length std	Standard deviation size of packets in the forward direction
bwd packet length max	Maximum size of packets in the backward direction
bwd packet length min	Minimum size of packets in the backward direction
bwd packet length mean	Mean size of packets in the backward direction
bwd packet length std	Standard deviation size of packets in the backward direction
Flow bytes/s	Flow bytes rate that is the number of bytes transferd per second
Flow packets/s	Flow packets rate that is the number packets transferd per second
Flow IAT Mean	the average time between two flows
Flow IAT std	Standard deviation time between two flows
Flow IAT Max	Maximum time between two flows
Flow IAT Min	Minimum time between two flows
Fwd IAT total	Total time between two packets sent in the forward direction
Fwd IAT mean	the mean time between two packets sent in the forward direction
Fwd IAT std	Standard deviation time between two packets sent in the forward direction
Fwd IAT max	Maximum time between two packets sent in the forward direction
Fwd IAT min	Minimum time between two packets sent in the forward direction
Bwd IAT total	total time between two packets sent in the backward direction
Bwd IAT min	Minimum time between two packets sent in the backward direction

Bwd IAT max	Maximum time between two packets sent in the backward direction
Bwd IAT mean	Mean time between two packets sent in the backward direction
Fwd PSH Flags	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
Bwd PSH Flags	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
Fwd URG Flags	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
Bwd URG Flags	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
Fwd Header Length	Total bytes used for headers in the forward direction.
Bwd Header Length	Total bytes used for headers in the forward direction
Fwd Packets/s	Number of forwarding packets per second
Bwd Packets/s	Number of backward packets per second
Packet Length Min	Minimum length of a flow
Packet Length Max	The maximum length of a flow
Packet Length Mean	Mean length of a flow
Packet Length Std	Standard deviation length of a flow
Packet Length Variance	Minimum inter-arrival time of packet
FIN Flag Count	Number of packets with FIN
SYN Flag Count	Number of packets with SYN
RST Flag Count	Number of packets with RST
PSH Flag Count	Number of packets with PUSH
ACK Flag Count	Number of packets with ACK
URG Flag Count	Number of packets with URG
CWE Flag Count	Number of packets with CWE
ECE Flag Count	Number of packets with ECE
Down/Up Ratio	Download and upload ratio
Average Packet Siz	The average size of packets
Fwd Segment Size Avg	Average size observed in the forward direction
Bwd Segment Size Avg	Average size observed in the backward direction
Fwd Bytes/Bulk Avg	The average number of bytes bulk rate in the forward direct
Fwd Bulk Rate Avg	The average number of bulk rate in the forward direction
Bwd Bytes/Bulk Avg	The average number of bytes bulk rate in the backward direction
Bwd Packet/Bulk Avg	The average number of packets bulk rate in the backward direction
Bwd Bulk Rate Avg	The average number of bulk rate in the backward direction
Subflow Fwd Packet	The average number of packets in a sub-flow in the forward direction
Subflow Fwd Bytes	The average number of bytes in a sub-flow in the forward direction
Subflow Bwd Packets	The average number of packets in a sub-flow in the backward direction
Subflow Bwd Bytes	The average number of bytes in a sub-flow in the backward direction
FWD Init Win Bytes	Number of bytes sent in the initial window in the forward direction
Bwd Init Win Bytes	The number of bytes sent in the initial window in the backward direction
Fwd Act Data Pkts	The number of packets with at least 1 byte of TCP data payload in the forward direction
Fwd Seg Size Min	Minimum segment size observed in the forward direction

Active Mean	The mean time a flow was active before becoming idle
Active Std	Standard deviation time a flow was active before becoming idle
Active Max	The maximum time a flow was active before becoming idle
Active Min	The minimum time a flow was active before becoming idle
Idle Mean	Meantime a flow was idle before becoming active
Idle Std	Standard deviation time a flow was idle before becoming active
Idle Max	The maximum time a flow was idle before becoming active
Idle Min	The minimum time a flow was idle before becoming active

Data Pre-Processing and Feature Engineering

Data pre-processing is a crucial step that ensures the quality and suitability of data for model training. This involves cleaning the data by handling missing values and outliers, normalizing data to bring all features to a similar scale, and encoding categorical variables into a format that can be used by machine learning algorithms. Feature engineering, on the other hand, involves creating new features from the existing ones, which could enhance the model's predictive ability. For instance, in our study, the dataset provided was imbalance so we had to balance the data ensuring both benign and DDoS attack data types were of same size taking only the first 40,000 sets for our paper. For feature engineering since we dealing with a new paradigm of networking we formulated the following new features for our paper.

Network Metrics Used

The following network metrics were used for detecting DDoS attacks

Speed of Source IP

This feature gives the number of source IPs per unit of time

$$SSIP = \frac{\text{SumIPsrc}}{T}$$

where SumIPsrc is the total number of IP sources incoming in every flow and T is the sampling time intervals. The time interval T is set to three seconds such that the detection system monitors and collects data of flows every five seconds and stores the number of source IPs during this duration. The controller needs to have sufficient data of both normal and attack traffic for the machine learning algorithm to predict the attacks. For normal attacks the SSIP is usually low and for attack the count is usually higher.

Speed of Flow Entries

This is the total number of flow entries to the switch in the network within a particular time interval. This is a very relevant parameter of attack detection because the number of flows increases significantly in a fixed interval of time in case of an attack as compared to the SFE value in times of normal traffic flows

$$\text{Speed of Flow Entries} = \frac{N}{T}$$

Standard Deviation of Flow Packets

This is the standard deviation of the number of packets in the T period. Abbreviated as SDFP, it is defined as

$$SDFP = \sqrt{\frac{1}{n} \sum_{i=1}^n (packet_i - mean\ packets)^2}$$

where $packet_i$ is the number of packets in the i th flow and $MeanPackets$ is the average number of packets in the T period in the network. This feature has high correlation to the event of a DDoS attack because in the case of an attack, the attacker sends large number of attack packets whose size is relatively small and this will have much lower standard deviation than the normal data packets resulting in a significant drop in this parameter during a DDoS attack.

Standard Deviation of Flow Bytes

This is the standard deviation of the number of bytes in the T period. Abbreviated as SDFB

$$SDFB = \sqrt{\frac{1}{n} \sum_{i=1}^n (bytes_i - mean\ bytes)^2}$$

where $bytes_i$ is the number of bytes in i th flow and $MeanBytes$ is the average number of bytes in the time period in the network. Similar to SDFP, SDFB also has high correlation to the event of a DDoS attack and the expected value of this parameter is lower in case of an attack than in the case of normal traffic flows.

Ratio of Pair-Flow Entries

This is the number of flow entries in the switch which are interactive divided by the total number of flows in the T period.

$$Ratio\ of\ Flow\ pairs = \frac{InteractiveIP}{N}$$

where $InteractiveIP$ is the total number of interactive IPs in the flow and N is the total number of IPs. Under normal traffic conditions, the i th flow will have the same source IP as the destination IP of the j th flow and the j th flow will have the same source IP as the destination IP of the i th flow. This constitutes an interactive flow which will not be the case under DDoS attack. Under attack, flow entries to the destination host in time T increases sharply and the destination host is unable to respond to them. Thus there will be an abrupt decrease in the number of interactive flows as soon as the attack starts. The total number of interactive flows is divided by the total number of flows so as make this detection parameter scalable to the network under different operating conditions.

Using these parameters, we trained a various machine learning algorithms to classify the incoming traffic to a switch as normal or attack.

Model Development and Training

Model development begins with choosing a suitable machine learning algorithm based on the problem at hand and the nature of the data. For instance, if the goal is to predict whether a patient has a certain disease (a binary outcome), algorithms like logistic regression, decision trees, random forests, and support vector machines might be suitable. Training the model involves feeding the processed data into the chosen algorithm and allowing the algorithm to learn the relationship between features. We used Support vector machine, Decision tree algorithms to perform training on a train set of 80% and a testing set of 20%.

Network Specification

In carrying out this research work, the SDN framework, data plane has multiple node/hosts virtually created using mininet and these were all connected to the openflow switch which defines the SDN protocols and the openflow protocol communicates with the control plane of the framework. Control plane controls the data plane and the switches and defines rules and also monitors the network traffic flow, here Ryu controller was used as the controller which provided the programming capabilities and allowed us to control the routing operations in the network. The control plane was programmed using python as Ryu is a python based controller and uses a python based API to communicate with the application layer, which in our case is network traffic applications.

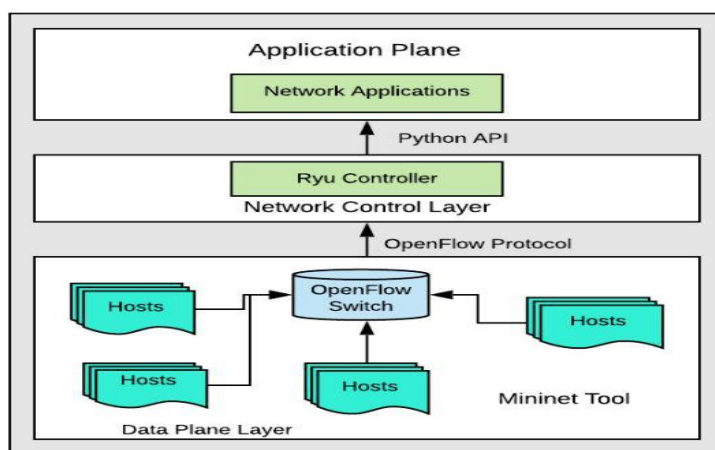


Figure 9: Network Specification (Kumar, 2020)

Network Design

The network topology was designed using mininet network simulator, the network has 5 hosts/nodes and one single openflow switch and one RYU controller. All the hosts were connected to the switch and the switch was connected to the controller. All of these hosts and switch were controlled by the Ryu controller.

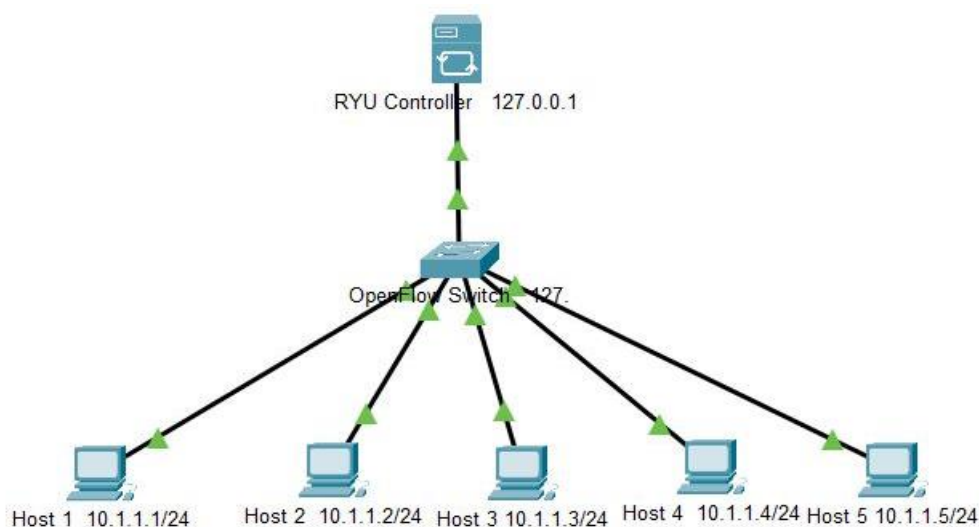


Figure 10: Network Topology (Testbed for Our Work)

Python Codes for the Creation of Our Network

```
def build(self):
    s1 = self.addSwitch('s1')#,dpid='0000000000002203')
    h1 = self.addHost('h1', ip='10.1.1.1/24', mac="00:00:00:00:00:01", defaultRoute="via
10.1.1.10")
    h2 = self.addHost('h2', ip='10.1.1.2/24', mac="00:00:00:00:00:02", defaultRoute="via
10.1.1.10")
    h3 = self.addHost('h3', ip='10.1.1.3/24', mac="00:00:00:00:00:03", defaultRoute="via
10.1.1.10")
    h4 = self.addHost('h4', ip='10.1.1.4/24', mac="00:00:00:00:00:04", defaultRoute="via
10.1.1.10")
    h5 = self.addHost('h5', ip='10.1.1.5/24', mac="00:00:00:00:00:05", defaultRoute="via
10.1.1.10")
    self.addLink(h1, s1, cls=TCLink, bw=10)
    self.addLink(h2, s1, cls=TCLink, bw=10)
    self.addLink(h3, s1, cls=TCLink, bw=10)
    self.addLink(h4, s1, cls=TCLink, bw=10)
    self.addLink(h5, s1, cls=TCLink, bw=10)
```

Figure 11: Network Creation with Python

Attack Detection Steps

In our SDN network environment the following steps were taken to collect network traffic and then classify it as attack or benign.

START:

Step 1: Initialized the SDN Controller and establish communication with the network devices on Mininet.

Step 2. Load the pre-trained machine learning models, such as SVM, Decision Trees

Step 3. Monitor Real-time Network Traffic:

Continuously captured and monitored real-time network traffic within the SDN environment.

Collected packet headers or flow-level information from incoming network packets.

Step 4 : Extract Relevant Features:

Extracted relevant features from the captured network traffic data.

Step 5. Perform Attack Detection:

Fed the extracted features into the loaded machine learning models.

Utilized the models to classify the network traffic as normal or malicious.

Step 6. Mitigation Actions:

If an attack is detected:

Trigger appropriate mitigation actions; Blocking ports and Dropping the packages

Step 7: Repeat Steps 3-6:

Continuously monitor, extract features, perform attack detection, and trigger mitigation actions in an iterative manner as new network traffic arrives.

Figure 11 depicts the view of how the process was performed

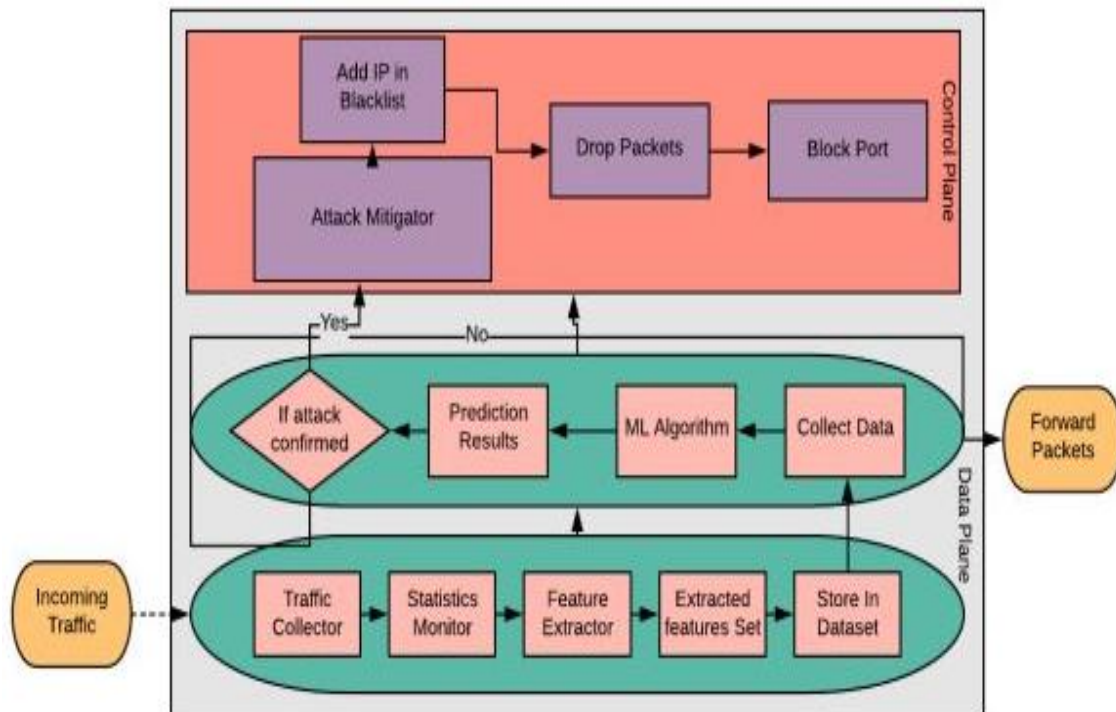
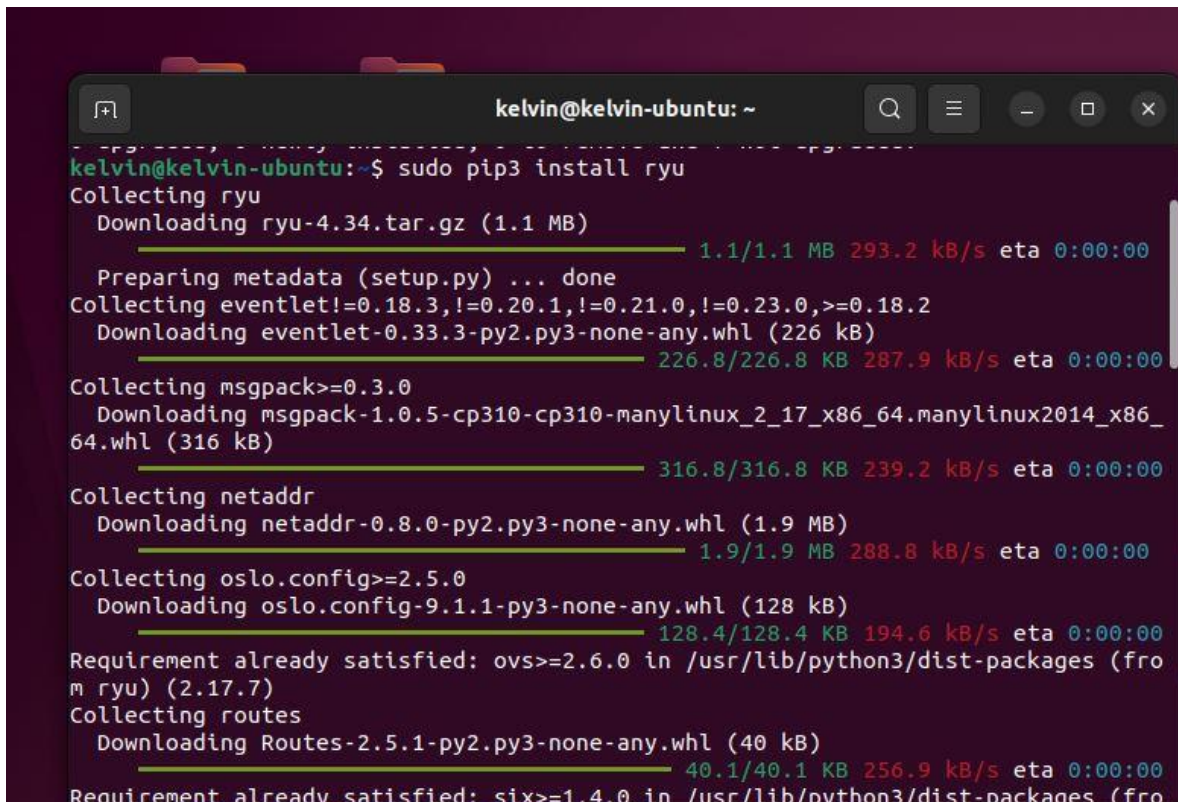


Figure 12: Steps Taken in SDN Environment to Detect Attack (Kumar, 2020)

Software defined networks has various protocols and controllers, each of those are designed to perform in a certain way and provide efficiency and flexibility in a particular aspect. The presented method is implemented using the most popular and out performing tools for the detection and mitigation of attacks in a software defined network. Openflow Protocol is the most popular and standard protocol for software defined networks, hence openVswitch is used for this project. As the presented method is a combination of statistical and machine learning methods, the logic and techniques are programmed using python. Statistical method includes parameters such as speed of source IP, speed of flow entries and ratio of flow pair entries, all of these logic is programmed in the controller.

Ryu Controller is an open-source python based programmable controller, which is used to define the rules and logic for the switches to follow in the methodology.



```
kelvin@kelvin-ubuntu: ~$ sudo pip3 install ryu
Collecting ryu
  Downloading ryu-4.34.tar.gz (1.1 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 293.2 kB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting eventlet!=0.18.3,!=0.20.1,!=0.21.0,!=0.23.0,>=0.18.2
  Downloading eventlet-0.33.3-py2.py3-none-any.whl (226 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 226.8/226.8 KB 287.9 kB/s eta 0:00:00
Collecting msgpack>=0.3.0
  Downloading msgpack-1.0.5-cp310-cp310-manylinux_2_17_x86_64_manylinux2014_x86_64.whl (316 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 316.8/316.8 KB 239.2 kB/s eta 0:00:00
Collecting netaddr
  Downloading netaddr-0.8.0-py2.py3-none-any.whl (1.9 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 288.8 kB/s eta 0:00:00
Collecting oslo.config>=2.5.0
  Downloading oslo.config-9.1.1-py3-none-any.whl (128 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 128.4/128.4 KB 194.6 kB/s eta 0:00:00
Requirement already satisfied: ovs>=2.6.0 in /usr/lib/python3/dist-packages (from ryu) (2.17.7)
Collecting routes
  Downloading Routes-2.5.1-py2.py3-none-any.whl (40 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40.1/40.1 KB 256.9 kB/s eta 0:00:00
Requirement already satisfied: six>=1.4.0 in /usr/lib/python3/dist-packages (from ryu) (1.16.0)
```

Figure 13: RYU Controller Installation

Mininet is a network simulator and creates a virtual network topology with controller, switches and hosts, in this work a single openVswitch with 1 switch and 5 hosts for the purpose of this work

Attack Generation

Hping3 is a packet generator which generates TCP/IP traffic in the network, it is mostly used to test network security. Normal and attack traffic scripts are written to generate traffic automatically using this tool. In this research work we deployed this tool exclusively for the purpose of attack generation.

Below is the sample code for attack used in this project

```
do
ping -c1 10.1.1.5
#TCP SYN flood, , and ICMP flood
hping3 --rand-source -i u10000 -S -d 64 -c 1000 10.1.1.5
#UDP flood
hping3 -2 --rand-source -i u15000 -d 64 -c 1000 10.1.1.5
#ICMP flood
hping3 -1 --rand-source -i u20000 -d 64 -c 1000 10.1.1.5
#TCP SYN flood, , and ICMP flood
hping3 --rand-source -i u25000 -S -d 64 -c 1000 10.1.1.5
#UDP flood
hping3 -2 --rand-source -i u30000 -d 64 -c 1000 10.1.1.5
#ICMP flood
hping3 -1 --rand-source -i u 35000 -d 64 -c 1000 10.1.1.5
done
```

```
do
ping -c1 10.1.1.5
#TCP SYN flood, , and ICMP flood
hping3 --rand-source -i u10000 -S -d 64 -c 1000 10.1.1.5
#UDP flood
hping3 -2 --rand-source -i u15000 -d 64 -c 1000 10.1.1.5
#ICMP flood
hping3 -1 --rand-source -i u20000 -d 64 -c 1000 10.1.1.5
#TCP SYN flood, , and ICMP flood
hping3 --rand-source -i u25000 -S -d 64 -c 1000 10.1.1.5
#UDP flood
hping3 -2 --rand-source -i u30000 -d 64 -c 1000 10.1.1.5
#ICMP flood
hping3 -1 --rand-source -i u 35000 -d 64 -c 1000 10.1.1.5
done
```

Materials Used

Here, we highlighted all the materials used in the implementation of this project. This included both software materials ranging from programming language tools, IDE, libraries, dataset and the hardware materials.

Software and library Materials

- i. Python Programming Language
- ii. Google Colab IDE (Integrated developer's environment)
- iii. Pandas
- iv. Numpy
- v. Matplotlib
- vi. Sklear
- vii. Seaborn
- viii. Mininet
- ix. Oracle VM Virtual box 7.0

Hardware Materials

For the process of design of the platform, a Computer system was used with the following characteristics:

- i. A Dell Latitude 5289
- ii. Processor Intel(R) Core (TM) i5-7300U @2.60GHz (4 CPUs)
- iii. Random Access Memory: 8.0 GB
- iv. Operating System: Windows 11 Pro 64-bits (10.0, Build 21996)
- v. Ubuntu 20.04
- vi. A modem for internet connection

3.0 FINDINGS

Detection Performance

To evaluate the performance of machine learning algorithms for attack detection in Software Defined Networking (SDN), we conducted experiments using the Cicddos2019 dataset, which has been widely used in previous research on Software Defined Networking environment. We trained and tested two popular machine learning algorithms, Support Vector Machines (SVM) and Decision Trees, using the CiCDDoS2019 dataset. The performance of each algorithm was assessed based on following evaluation metrics; accuracy, precision, recall, and F1-score

Table 4: Performance Metrics of ML Algorithms

Algorithm	Accuracy	Precision	Recall	F1-Score
Decision Tree	88%	84%	87%	89%
SVM	93%	92%	94%	93%

The results indicate that both SVM and Decision Trees achieved high accuracy in detecting attacks in SDN. SVM achieved an accuracy of 0.93, with a precision of 0.92, recall of 0.94, and an F1-score of 0.93. Similarly, Decision Trees achieved an accuracy of 0.88, with a precision of 0.84, recall of 0.87, and an F1-score of 0.89. Indicating these algorithms are suitable for implementation of a mechanism for attack detection and mitigation in SDN.

Network Metric Visualization

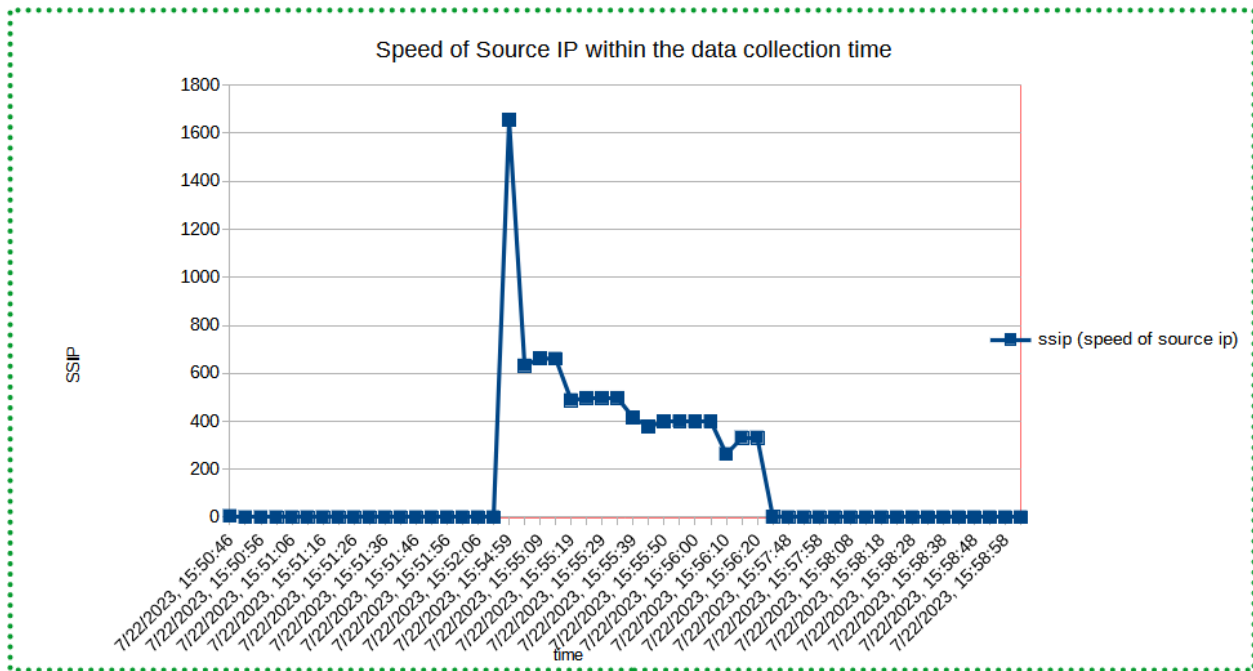


Figure 14: SSIP During Normal and Attack Session

From the above figure it show there is aspike in the number of new source IP during an attack.

Std Deviation of Flow Packages with time

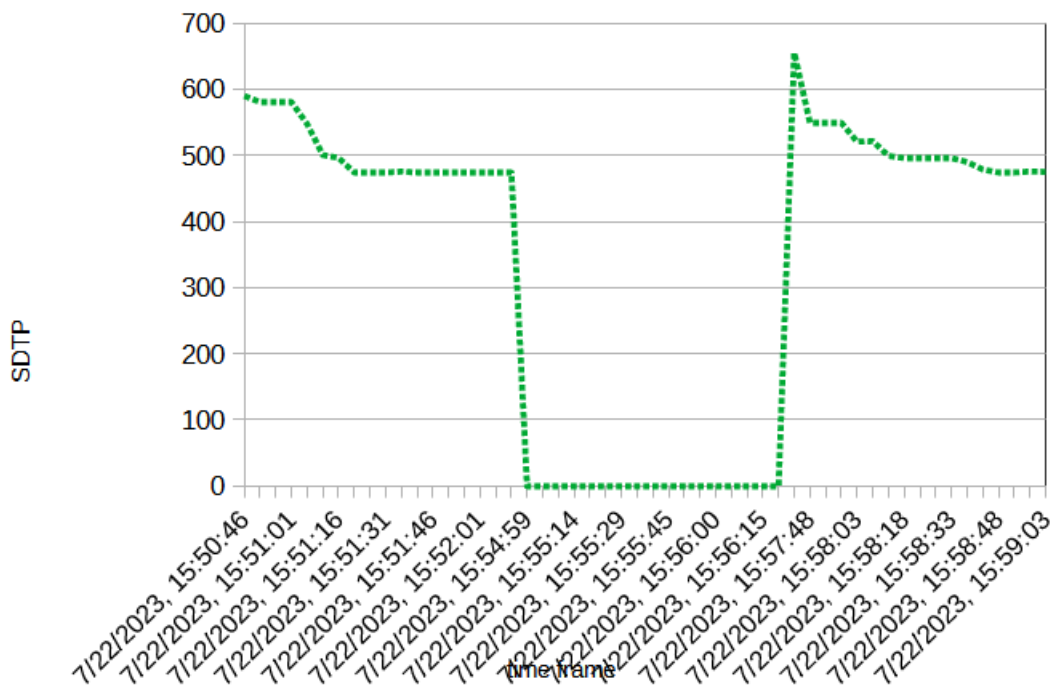


Figure 15: SDFP in Attack and Normal Session

The figure shows a lower standard of deviation in Flow packages during an attack opposed to a higher value during a normal session.

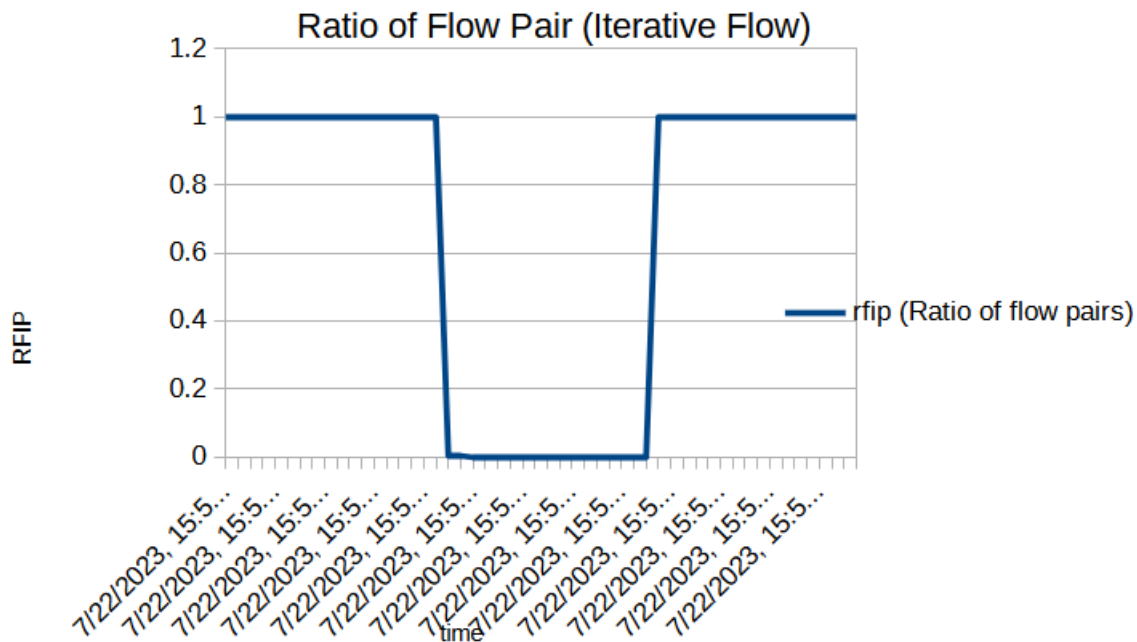


Figure 16: Ratio of Flow Pairs in Normal and Attack Session

The figure demonstrates a lower ratio of interactive flow during an attack as there are lesser Acknowledgement messages as the Controller is saturated and flooded.

Presentation of Results on the Software Defined Networking Environment

Our SDN network:

Using the command line shell on ubuntu, run the following command to create our network with mininet; 5 hosts and 1 server connected to the RYU controller

Sudo python3 topo.py

```
kelvin@kelvin:~/Desktop/mengproject/v1$ sudo python3 topo.py
Connecting to remote controller at 127.0.0.1:6653
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1
*** Adding links:
(10.00Mbit) (10.00Mbit) (h1, s1) (10.00Mbit) (10.00Mbit) (h2, s1) (1
0.00Mbit) (10.00Mbit) (h3, s1) (10.00Mbit) (10.00Mbit) (h4, s1) (10.
00Mbit) (10.00Mbit) (h5, s1)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
c1
*** Starting 1 switches
s1 ... (10.00Mbit) (10.00Mbit) (10.00Mbit) (10.00Mbit) (10.00Mbit)
```

Figure 17: Network Data Plane Implementation

Data Collection

On another shell, our controller was started using the following command

Ryu-manager app.py

Based on the selected mode, we will have the controller analyse the data from the switch and save it into result.csv file which will be used for validating our detection model.

```
kelvin@kelvin: ~/Desktop/me... x kelvin@kelvin: ~/Desktop/me... x kelvin@kelvin: ~/Desktop/me... x
kelvin@kelvin:~/Desktop/mengproject$ cd v1
kelvin@kelvin:~/Desktop/mengproject/v1$ ryu-manager app.py
loading app app.py
loading app ryu.controller.ofp_handler
instantiating app app.py of DDoSML
Our System has been launched in Data Collection State
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Figure 19: RYU Controller in Data Collection Mode; Control Plane

Visualizing Result File

Below is a picture of the result file which contains the calculated values for our network parameters as discussed in section 3.

It contain the speed of source ip, standard deviation of bytes, standard deviation of packets, ratio of flow pairs and the duration.

```
kelvin@kelvin: ~/Desktop/me... x kelvin@kelvin: ~/Desktop/me... x kelvin@kelvin: ~/Desktop/me... x
kelvin@kelvin:~/Desktop/mengproject/v1$ ls
analysis      ml.py          readme.txt      topo.py
app.py        normal.sh      result.csv
attack.sh     __pycache__   switch_1_data.csv
logic.txt     'raw data'    switch_1_flowcount.csv
kelvin@kelvin:~/Desktop/mengproject/v1$ tail result.csv
9,5,0.8888888888888888,0.0,0.0,0
1,0,1.0,528.7795591947765,1067346.610735238,0
2,0,1.0,463.75505388081757,905762.5720351484,0
2,0,1.0,430.84767584405023,825028.534104071,0
0,0,1.0,436.0425488362396,837679.3921291798,0
2,0,1.0,405.51233026876014,777454.5688968049,0
0,0,1.0,409.76576642434804,790280.5217942023,0
2,0,1.0,391.50332430411936,748477.410111437,0
kelvin@kelvin:~/Desktop/mengproject/v1$ tail -f result.csv
2,0,1.0,463.75505388081757,905762.5720351484,0
2,0,1.0,430.84767584405023,825028.534104071,0
0,0,1.0,436.0425488362396,837679.3921291798,0
2,0,1.0,405.51233026876014,777454.5688968049,0
0,0,1.0,409.76576642434804,790280.5217942023,0
2,0,1.0,391.50332430411936,748477.410111437,0
0,0,1.0,410.84761737885543,799832.3685737536,0
2,0,1.0,356.0630451272122,668033.1995015864,0
0 0 1 0 350 01808590887494 657613 2816437829 0
```

Figure 20: Visualizing the Values in Result File

Mimicking an Attack on Network

After creating our network, we can then change the parameters of our attack bash file using the hping3 tool to mimic an attack on the whole Mininet network.

```
kelvin@kelvin: ~/Desktop/me... x kelvin@kelvin: ~/Desktop/me... x kelvin@kelvin: ~/Desktop/me... x
kelvin@kelvin:~/Desktop/mengproject/v1$ sudo python3 topo.py
Connecting to remote controller at 127.0.0.1:6653
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1
*** Adding links:
(10.00Mbit) (10.00Mbit) (h1, s1) (10.00Mbit) (10.00Mbit) (h2, s1) (1
0.00Mbit) (10.00Mbit) (h3, s1) (10.00Mbit) (10.00Mbit) (h4, s1) (10.
00Mbit) (10.00Mbit) (h5, s1)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
c1
*** Starting 1 switches
s1 ... (10.00Mbit) (10.00Mbit) (10.00Mbit) (10.00Mbit) (10.00Mbit)
Potential Attack traffic generation.....
```

Figure 21: Mimicking Attack in the Data Plane (SDN Network)

The data can then be collected for the attack state as well and saved under the result.csv file as well.

Launching the RYU Controller in Detection Mode

Our ryu contronller can then be started in detection mode to ensure any traffic that passes through the switch to any part of the host devices is checked for potential attack or not.

```
kelvin@kelvin: ~/Desktop/mengproject/v1
kelvin@kelvin:~/Desktop/mengproject$ cd v1
kelvin@kelvin:~/Desktop/mengproject/v1$ ryu-manager app.py
loading app app.py
loading app ryu.controller.ofp_handler
instantiating app app.py of DDoSML
Our System has been launched in Attack Detection State
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Figure 22: Starting the RYU Controller in Detection

At this stage, the controller analyses any traffic in the network and creates the various parameters which are then passed through the machine learning detection model to detect for attack or not.

Analyzing the Traffic and Displaying the State of the Network

After the controller picks the traffic in the network, it classify the traffic based on the network parameters and from the model detection, it displays whether we have an attack or not.


```
kelvin@kelvin: ~/Desktop/mengproject/v1 x kelvin@kelvin: ~/Desktop/mengproject/v1 x
4992.9628062556
The Whole system is in Normal State; Thank You
sfe 4 ssip 0 rfip 0.8 sdfp 326.3359720696857 sdfb 592395.3183138585
The Whole system is in Normal State; Thank You
sfe 2 ssip 0 rfip 0.8333333333333334 sdfp 317.66529937064644 sdfb 57
6809.1215502803
The Whole system is in Normal State; Thank You
sfe 2 ssip 0 rfip 0.8571428571428571 sdfp 328.28669459201944 sdfb 59
3671.1123955107
The Whole system is in Normal State; Thank You
sfe 2 ssip 0 rfip 0.875 sdfp 301.9132049668138 sdfb 543123.058726043
6
The Whole system is in Normal State; Thank You
sfe 0 ssip 0 rfip 0.875 sdfp 309.7369246634957 sdfb 562193.566622727
7
The Whole system is in Normal State; Thank You
sfe 0 ssip 0 rfip 0.875 sdfp 309.5730864163313 sdfb 561431.831456678
2
The Whole system is in Normal State; Thank You
sfe 3 ssip 0 rfip 0.9473684210526315 sdfp 288.71439174381317 sdfb 51
7160.9771603296
The Whole system is in Normal State; Thank You
```

Figure 23: Analyzing the Network Traffic and Displaying the State of the Networking System

Starting a New Attack in the Mininet Shell

With our network up and active we can then use the hping3 tool to launch a new attack from host 1 to host 5 or any host per say and check the control plane of the RYU controller is the attack has been detected.

```
mininet>
mininet>
mininet> h1 hping3 -1 --rand-source -i u10000 -c 10000 10.1.1.5
PING 10.1.1.4 (10.1.1.4) 56(84) bytes of data.
64 bytes from 10.1.1.4: icmp_seq=1 ttl=64 time=0.813 ms

--- 10.1.1.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.813/0.813/0.813/0.000 ms
PING 10.1.1.5 (10.1.1.5) 56(84) bytes of data.
64 bytes from 10.1.1.5: icmp_seq=1 ttl=64 time=0.665 ms

--- 10.1.1.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.665/0.665/0.665/0.000 ms
Welcome to Our Attack Detection System 5 times
HPING 10.1.1.5 (h1-eth0 10.1.1.5): icmp mode set, 28 headers + 0 dat
a bytes
```

Figure 24: New Attack Using Hping 3 Tool

Attack Detection No Mitigation Engaged

At this stage the detection model is set to only detect the attack and display a warning message to the screen. At this point we can consider our system as an Intrusion detection system (IDS)

```
kelvin@kelvin: ~/Desktop/mengproject/v1
The whole system is in Normal State; Thank You
sfe 112 ssip 106 rfip 0.12903225806451613 sdfp 144.07590810434726 sd
fb 253192.58320354906
Switch 1 is experiencing a potential attack, Engaged in Mitigation
or Contact Security Team
sfe 458 ssip 458 rfip 0.027491408934707903 sdfp 69.00775985285559 sd
fb 125628.90021571699
Switch 1 is experiencing a potential attack, Engaged in Mitigation
or Contact Security Team
sfe 460 ssip 460 rfip 0.015355086372360844 sdfp 51.23084590416452 sd
fb 92950.69995820428
Switch 1 is experiencing a potential attack, Engaged in Mitigation
or Contact Security Team
sfe 462 ssip 461 rfip 0.011968085106382979 sdfp 43.25717898386274 sd
fb 78849.79429827734
Switch 1 is experiencing a potential attack, Engaged in Mitigation
or Contact Security Team
sfe 458 ssip 458 rfip 0.009174311926605505 sdfp 36.5271948580348 sdf
b 65663.2548226578
Switch 1 is experiencing a potential attack, Engaged in Mitigation
or Contact Security Team
sfe 462 ssip 462 rfip 0.007425742574257425 sdfp 32.99843906617625 sd
fb 59488.3299539812
Switch 1 is experiencing a potential attack, Engaged in Mitigation
```

Figure 25: Controller Detects an Attack

Controller Attack Mitigation

In this module, the controller was trained to detect the malicious attack and intent take a decision of blocking the ports of the attack source and also the packets of the attack.

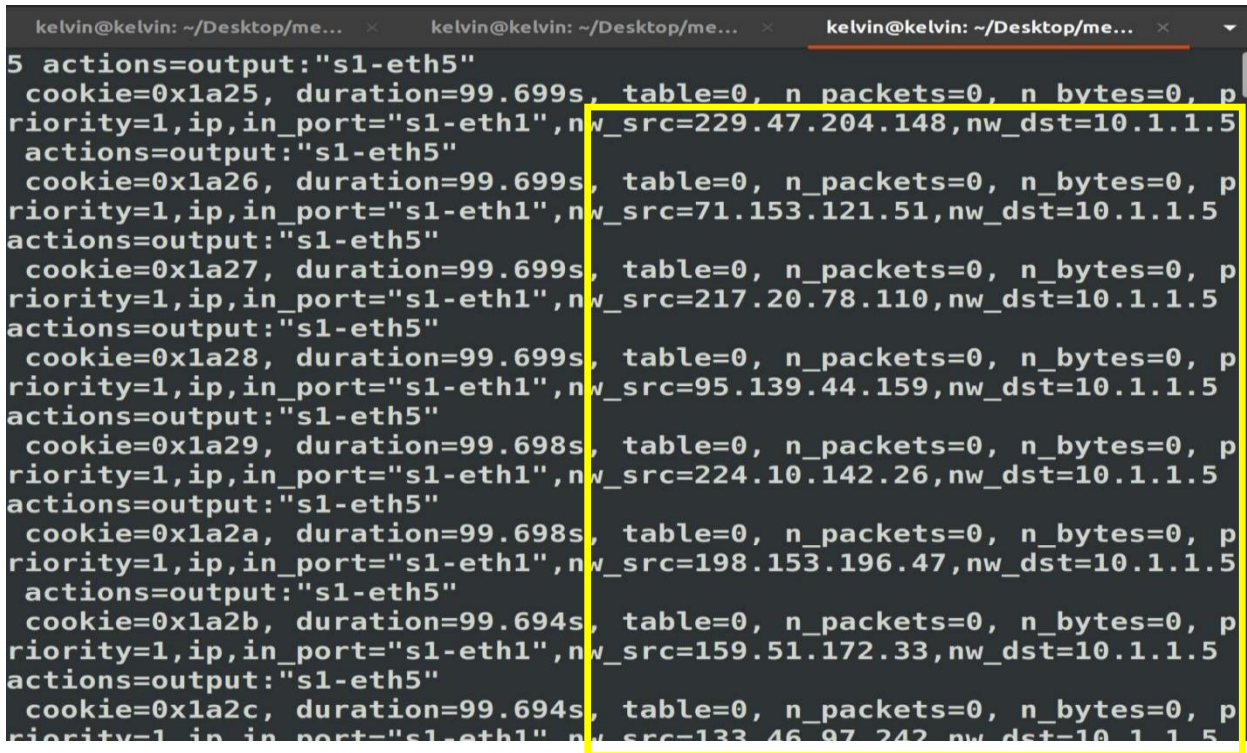
```
kelvin@kelvin: ~/Desktop/me...
The Whole system is in Normal State; Thank You
sfe 0 ssip 0 rfip 1.0 sdfp 279.08093525185956 sdfb 557204.2554174981
The Whole system is in Normal State; Thank You
sfe 0 ssip 0 rfip 1.0 sdfp 290.925288835829 sdfb 589988.5417123213
The Whole system is in Normal State; Thank You
sfe 0 ssip 0 rfip 1.0 sdfp 266.05062339811633 sdfb 526605.4142985399
The Whole system is in Normal State; Thank You
sfe 0 ssip 0 rfip 1.0 sdfp 251.96810052903626 sdfb 487006.1942192911
The Whole system is in Normal State; Thank You
sfe 116 ssip 116 rfip 0.14705882352941177 sdfp 100.13419726483875 sd
fb 181047.30944891408
Switch 1 is experiencing a potential attack, Engaged in Mitigation
or Console Security Team
System Engaged in Mitigation mode, Attack will be mitigated in a few
2023-08-19 22:35:10.645535 : Potential attack detected in switch 1
from port 3
2023-08-19 22:35:10.646024: Switch 1 Blocked the affected port 3
2023-08-19 22:35:10.646384: Switch 1 Removed the attacker flows wi
th potential harm
sfe -119 ssip -117 rfip 0.7058823529411765 sdfp 333.71778870776427 s
dfb 691225.9234484651
The Whole system is in Normal State; Thank You
```

Figure 26: Controller Mitigates Attack

After dropping the packets and the blocking the source of the attack, the system can then return to its normal state.

Visualizing the Flows in the Network During an Attack

Below is a screenshot showing the flows in our network when the system is under an attack.



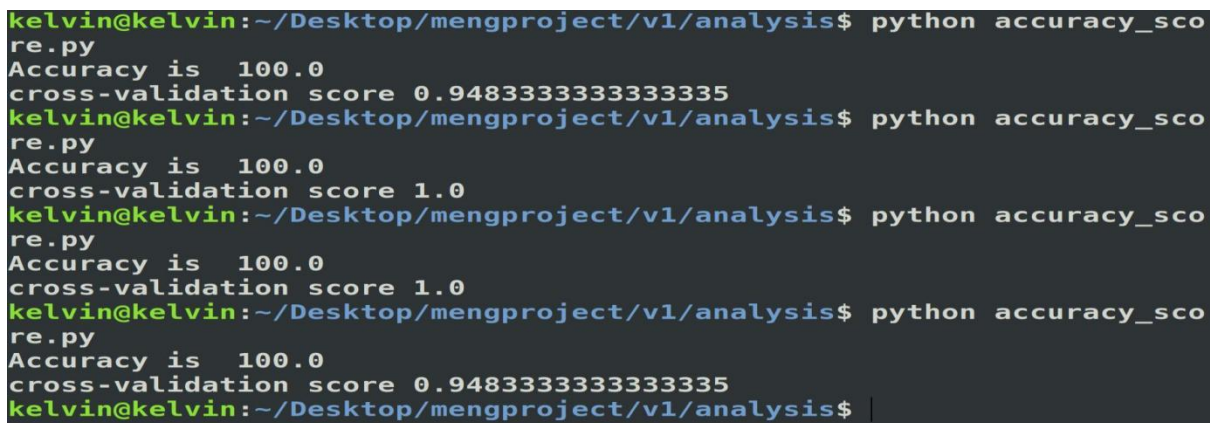
```
kelvin@kelvin: ~/Desktop/me... x kelvin@kelvin: ~/Desktop/me... x kelvin@kelvin: ~/Desktop/me... x
5 actions=output:"s1-eth5"
cookie=0x1a25, duration=99.699s, table=0, n_packets=0, n_bytes=0, p
riority=1, ip, in_port="s1-eth1", nw_src=229.47.204.148, nw_dst=10.1.1.5
actions=output:"s1-eth5"
cookie=0x1a26, duration=99.699s, table=0, n_packets=0, n_bytes=0, p
riority=1, ip, in_port="s1-eth1", nw_src=71.153.121.51, nw_dst=10.1.1.5
actions=output:"s1-eth5"
cookie=0x1a27, duration=99.699s, table=0, n_packets=0, n_bytes=0, p
riority=1, ip, in_port="s1-eth1", nw_src=217.20.78.110, nw_dst=10.1.1.5
actions=output:"s1-eth5"
cookie=0x1a28, duration=99.699s, table=0, n_packets=0, n_bytes=0, p
riority=1, ip, in_port="s1-eth1", nw_src=95.139.44.159, nw_dst=10.1.1.5
actions=output:"s1-eth5"
cookie=0x1a29, duration=99.698s, table=0, n_packets=0, n_bytes=0, p
riority=1, ip, in_port="s1-eth1", nw_src=224.10.142.26, nw_dst=10.1.1.5
actions=output:"s1-eth5"
cookie=0x1a2a, duration=99.698s, table=0, n_packets=0, n_bytes=0, p
riority=1, ip, in_port="s1-eth1", nw_src=198.153.196.47, nw_dst=10.1.1.5
actions=output:"s1-eth5"
cookie=0x1a2b, duration=99.694s, table=0, n_packets=0, n_bytes=0, p
riority=1, ip, in_port="s1-eth1", nw_src=159.51.172.33, nw_dst=10.1.1.5
actions=output:"s1-eth5"
cookie=0x1a2c, duration=99.694s, table=0, n_packets=0, n_bytes=0, p
riority=1, ip, in_port="s1-eth1", nw_src=133.46.97.242, nw_dst=10.1.1.5
```

Figure 27: Traffic Visualization during an Attack Session

We noticed a good number of new source ip in the network flow during an attack.

Model Evaluation on Mininet Environment

The accuracy score of the model on the system based on attack detection.



```
kelvin@kelvin:~/Desktop/mengproject/v1/analysis$ python accuracy_score.py
Accuracy is 100.0
cross-validation score 0.9483333333333335
kelvin@kelvin:~/Desktop/mengproject/v1/analysis$ python accuracy_score.py
Accuracy is 100.0
cross-validation score 1.0
kelvin@kelvin:~/Desktop/mengproject/v1/analysis$ python accuracy_score.py
Accuracy is 100.0
cross-validation score 1.0
kelvin@kelvin:~/Desktop/mengproject/v1/analysis$ python accuracy_score.py
Accuracy is 100.0
cross-validation score 0.9483333333333335
kelvin@kelvin:~/Desktop/mengproject/v1/analysis$
```

Figure 28: Metric Evaluation on the Mininet SDN Network

Figure 28 showed accuracy score and cross validation rate for Support vector machine, Decision tree, performed at different times.

Discussion

The methodology used in this project was aimed at solving the problem faced by Controllers deployed in Software defined Networking environments. key objectives were to design a SDN network, create network flows which mimic both attack and mitigation, deploy a machine learning mechanism to

detect and mitigate these flows in real time. We effectively deployed this mechanism in a Mininet emulator environment which showed great results in the detection and mitigation of attacks in SDN as shown in the results above.

In traditional networking environments, there exist firewalls, IPS like Snort, Suricata and many others which exist but with limited functionalities as they function based on signatures stored in their security database. We developed and used a Python-based machine learning approach and mechanism for analyzing the network traffic. The training was carried out on CiCDDoS2019 dataset. Feature selection was performed and we created new network metrics for the detection of attack; Speed of Source IP, Speed of Flow entries, Standard Deviation of flow packets, Standard deviation of Flow entries and Ratio of flow pairs. The results demonstrate the effectiveness of the machine learning-based attack mitigation techniques in reducing attack traffic and improving network performance. The dynamic flow control and reconfiguration technique achieved a significant reduction of 93% in attack traffic detection, these findings align (Amrish et al., 2022) that have also utilized machine learning algorithms for attack mitigation in SDN. The results indicate that machine learning-based techniques can effectively mitigate the impact of attacks and optimize network performance in SDN environments.

Our mechanism showed great mitigation strategy by dropping and blocking affected ports which aligned with (Kousar et al., 2021) study on dynamic flow monitoring and adjustment, this technique involves dynamically adjusting flow rules in the network to divert or drop traffic associated with detected attacks. By reconfiguring the network flow paths, malicious traffic can be isolated or redirected to mitigate the impact on legitimate traffic.

While the results of this research demonstrated the effectiveness of machine learning algorithms for enhancing attack detection and mitigation in SDN, there are some limitations that need to be acknowledged. First, the performance of the machine learning algorithms heavily relied on the quality and representativeness of the training dataset. The CiCDDoS2019 dataset used in this study, while widely used in previous research, may not cover all possible attack scenarios and network environments with matches same issues (Sharafaldin et al., 2019) discussed in their work and reason they engaged in the creation of the CiCDDoS2019 dataset.

Overall, the results and discussion highlight the potential of machine learning algorithms, particularly SVM and Decision Trees, in enhancing attack detection and mitigation in SDN. The research provides insights into their performance, impact on network performance, and a guide for future research and implementations in real world enterprise networks.

4.0 CONCLUSIONS AND RECOMMENDATIONS

Conclusion

In this study we explored the capabilities of integrating machine learning models to effectively detect and mitigate attacks in Software Defined Networking Environment. Software defined network provides us the capabilities to design and perform operations in the network by programming which is not case with traditional networks. While SDN is changing the networking industry and a promising future technology it currently has mostly research applications and very few industry applications by the major players. With the control plane of the SDN network resting in the controller, it becomes so easy for attackers to target the controller and take control of the entire network. Protecting the controller by detecting and mitigating against any intrusion was the objective of this research. The research and development efforts in enhancing attack detection and mitigation in SDN using machine learning are crucial for addressing the evolving landscape of network security threats. By harnessing the power of machine learning algorithms, SDN can become more intelligent, adaptive, and capable of defending against sophisticated attacks. As the field of machine learning and SDN continues to

advance, further research and collaboration are needed to overcome the challenges and unlock the full potential of this technology in securing our networks.

The implemented method is a combination of statistical features; speed of source IP, speed of flow entries, flow count and ratio of flow-pair and machine learning algorithm to detect and predict DDOS attacks in the network, experimented results shows the presented method provided high accuracy, higher detection rate with lesser false predictions.

Recommendations

Based on the findings and knowledge acquired, some key recommendations for successful implementation of an Enhanced attack and detection scheme in SDN include:

- i. Use deep learning and Ensemble learning as the system will have an awareness of its state and hence have better accuracy and less false alarm rates.
- ii. Conducting thorough feature analysis and selection based on statistical techniques, correlation analysis, and domain knowledge.
- iii. Experimenting with multiple algorithms like Deep Neural Networks, Ensemble Learning algorithms.
- iv. Optimizing the system to minimize computational overhead and ensure real-time processing.
- v. Performing the study on a Real world SDN environment to ensure proper knowledge of the data flow patterns in real world environments.
- vi. Use multiple datasets in the implementation of the system.

Contribution

This research project has contributed significantly in advancing knowledge by providing a mechanism that can be deployed to secure real world networks of various companies especially those deployed through the concept of Software defined networking in detecting and mitigating attacks in real-time. Providing new ways of implementing intrusion detection and intrusion prevention systems using the mechanism of machine learning. Our approach has the ability to determine normal traffic from abnormal traffic and to detect and attack in a system in real time with high accuracy and low-rate false alarms.

REFERENCES

- Aksu, Doğukan & Ustebay, Serpil & Aydin, M.Ali & Atmaca, Tülin. (2018). Intrusion Detection with Comparative Analysis of Supervised Learning Techniques and Fisher Score Feature Selection Algorithm. 10.1007/978-3-030-00840-6_16.
- Amrish, R., Bavapriyan, K., Gopinaath, V., Jawahar, A., & Kumar, C. V. (2022). DDoS detection using machine learning techniques. *Journal of IoT in Social, Mobile, Analytics, and Cloud*, 4(1), 24-32.
- Bawany, N. Z., Shamsi, J. A., & Salah, K. (2017). DDoS attack detection and mitigation using SDN: methods, practices, and solutions. *Arabian Journal for Science and Engineering*, 42, 425-441.
- Bergstra, J., Yamins, D., & Cox, D. D. (2013, June). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference* (Vol. 13, p. 20).
- Cameron Magazine (2021). <https://www.cameroonmagazine.com/actualite-internationale/cm-software-defined-networking-sdn-market-development-strategies-growth-rate-and-opportunity-assessment-till-2025/>
- C. Koliass, G. Kambourakis, A. Stavrou, and J. Voas, 2017 “DDoS in the IoT: Mirai and other botnets,” *Computer* (Long Beach Calif), vol. 50, no. 7, doi: 10.1109/MC.2017.201
- Carlos Javier, Gonzalez. (2017). Management of a heterogeneous distributed architecture with the SDN.
- Chan P & Vargiya, R. (2013). Boundary Detection in Tokenizing Network Application Payload for Anomaly Detection. Melbourne: Florida Institute of Technology
- Chen, W.-K., 1993. *Linear Networks and Systems*. Wadsworth, Belmont, CA, USA, pp. 123–135.
- Chirag N. Modi, Dhiren R. Patel, Avi Patel, et al. “Bayesian Classifier and Snort based network intrusion detection system in cloud computing”. In: 2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT’12). ISSN: null. July 2012, pp. 1–7. doi: 10 . 1109 / ICCCNT.2012.6396086.
- DDoS attack that disrupted internet was largest of its kind in history, experts say | Hacking | The Guardian.” <https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet> (accessed May . 10, 2023).
- Eliyan, L.F.; Di Pietro, (2021). DoS and DDoS attacks in Software Defined Networks: A survey of existing solutions and research challenges. *Future Gener. Comput. Syst.*
- Elsayed, M. S., Le-Khac, N. A., Dev, S., & Jurcut, A. D. (2019, October). Machine-learning techniques for detecting attacks in SDN. In *2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)* (pp. 277-281). IEEE.
- Farhady, H., Lee, H., & Nakao, A. (2015). Software-defined networking: A survey. *Computer Networks*, 81, 79-95.
- Garg, S., Kaur, K., Kumar, N., Kaddoum, G., Zomaya, A. Y., & Ranjan, R. (2019). A hybrid deep learning-based model for anomaly detection in cloud datacenter networks. *IEEE Transactions on Network and Service Management*, 16(3), 924-935.
- Gueant, V. (2021). iPerf-iPerf3 and iPerf2 user documentation. *Iperf. fr*.
- Gong, D. F. (2003). Deciphering Detection Techniques: Part II Anomaly-Based Intrusion Detection. McAfee Security.

- Hande, Y., & Muddana, A. (2019, November). Intrusion detection system using deep learning for software defined networks (SDN). In *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT)* (pp. 1014-1018). IEEE.
- Hardesty, L. (2017). Google Brings SDN to the Public Internet. sdxcentral: <https://www.sdxcentral.com/articles/news/google-brings-sdn-publicinternet/2017/04/>, Access Date:21.04.2023
- Hinden, R. M. (2014). SDN And Security: Why take over the hosts while you can take the whole network. RSA Conference: Capitalizing on collective intelligence. San Francisco.
- Hussain, J., & Hnamte, V. (2021, September). A novel deep learning based intrusion detection system: Software defined network. In *2021 International Conference on innovation and intelligence for informatics, computing, and technologies (3ICT)* (pp. 506-511). IEEE.
- Islam, Md Tariqul & Islam, Nazrul & Refat, Md. (2020). Node to Node Performance Evaluation through RYU SDN Controller. *Wireless Personal Communications*. 112. 10.1007/s11277-020-07060-4.
- “Isms family of standards,” standard, International Organization for Standardization, Geneva, CH, 2018
- Jammal, M., Singh, T., Shami, A., Asal, R., & Li, Y. (2014). Software defined networking: State of the art and research challenges. *Computer Networks*, 72, 74-98.
- Janabi, A. H., Kanakis, T., & Johnson, M. (2022). Convolutional neural network based algorithm for early warning proactive system security in software defined networks. *IEEE Access*, 10, 14301-14310.
- Javaid, A., Niyaz, Q., Sun, W., (2016, May). A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)* (pp. 21-26).
- Jafarian, Tohid & Masdari, Mohammad & Ghaffari, Ali & Majidzadeh, Kambiz. (2020). Security anomaly detection in software-defined networking based on a prediction technique. *International Journal of Communication Systems*. 33. e4524. 10.1002/dac.4524.
- Junhong T (2020) A Machine Learning Framework for Host Based Intrusion Detection using machine learning.
- Kanakarajan, N. K., & Muniyasamy, K. (2016). Improving the accuracy of intrusion detection using gar-forest with feature selection. In *Proceedings of the 4th International Conference on Frontiers in Intelligent Computing: Theory and Applications (FICTA) 2015* (pp. 539-547). Springer India.
- Kaur, Sukhveer & Singh, Japinder & Ghumman, Navtej. (2014). Network Programmability Using POX Controller. 10.13140/RG.2.1.1950.6961.
- Kolias, N., Moustafa, N., & Sitnikova, E. (2017). Forensics and deep learning mechanisms for botnets in internet of things: A survey of challenges and solutions. *IEEE Access*, 7, 61764-61785.
- Kousar, H., Mulla, M. M., Shettar, P., & Narayan, D. G. (2021, June). Detection of DDoS attacks in software defined network using decision tree. In *2021 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT)* (pp. 783-788). IEEE.
- Kurochkin, I. I., & Volkov, S. S. (2020, September). Using GRU based deep neural network for intrusion detection in software-defined networks. In *IOP Conference Series: Materials Science and Engineering* (Vol. 927, No. 1, p. 012035). IOP Publishing.

- Kumar Singh, V. (2020). DDOS attack detection and mitigation using statistical and machine learning methods in SDN (Doctoral dissertation, Dublin, National College of Ireland).
- Labonne, M., Olivereau, A., Polvé, B., & Zeglache, D. (2019, January). A cascade-structured meta-specialists approach for neural network-based intrusion detection. In *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)* (pp. 1-6). IEEE
- Le J (2017). A logitboost-based algorithm for detecting known and unknown attacks in Networks
- Lemaître, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *The Journal of Machine Learning Research*, 18(1), 559-563.
- Leung, K., & Leckie, C. (2005, January). Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38* (pp. 333-342).
- Lim, A. (2015, July). Security risks in SDN and other new software issues. In *RSA Conference*.
- M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, 2007, Ethane: Taking Control of the Enterprise in Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ser. SIGCOMM 07. ACM
- Martin Casado (December 2007). "[Architectural Support for Security Management in Enterprise Networks](#)" (PDF). PhD dissertation. Stanford University. Retrieved October 30, 2016
- Mell, P. (2007). Intrusion detection and prevention systems. In *Handbook of Information and Communication Security* (pp. 177-192). Berlin, Heidelberg: Springer Berlin Heidelberg.
- Metzler, J. (2014). SDN and Network Virtualization: A Reality Check. *Network World*: <https://www.networkworld.com/article/2604023/software-definednetworking/sdn-and-network-virtualization-a-reality-check.html>, Access date:6.05.2023.
- Mittal, Sangeeta. (2018). Performance Evaluation of Openflow SDN Controllers. 10.1007/978-3-319-76348-4_87.
- Modi, M., Abd Allah, M., & Tawfik, B. (2012). Intrusion detection model using naive bayes and deep learning technique. *Int. Arab J.*
- Myint oo, S., & Kaur, G. (2019). SVM Implementation for DDoS Attacks in Software Defined Networks. *International Journal of Innovative Technology and Exploring Engineering*
- Nakandala, S., Zhang, Y., & Kumar, A. (2020). Cerebro: A data system for optimized deep learning model selection. *Proceedings of the VLDB Endowment*, 13(12), 2159-2173.
- Neupane, R.L., Neely, T., Chetri, N., Vassell, M., Zhang, Y., Calyam, P., Durairajan, R.: Dolus, 2018. In: Proceedings of the 19th International Conference on Distributed Computing and Networking - ICDCN '18. pp. 1–10. ACM Press, New York, New York, USA
- "NSL-KDD | Datasets | Research | Canadian Institute for Cybersecurity | UNB."
<https://www.unb.ca/cic/datasets/nsl.html> (accessed May 07, 2023).
- Nunez, A., Ayoka, J., Islam, M. Z., & Ruiz, P. (2023). A Brief Overview of Software-Defined Networking. *arXiv preprint arXiv:2302.00165*.
- P. FARINA, E. CAMBIASO, G. PAPAEO and M. AIELLO, 2015 "Understanding DDoS Attacks from Mobile Devices" 3rd International Conference on Future Internet of Things and Cloud, Rome

- Preprocessing data — scikit-learn 0.22.2 documentation. retrieved: <https://scikit-learn.org/stable/modules/preprocessing.html> (visited on 03/06/2023)
- R. T. Kokila, S. T. Selvi and K. Govindarajan 2014, “DDoS detection and analysis in SDN-based environment using support vector machine classifier,” IEEE sixth international conference on advanced computing (ICoAC) (pp. 205-210).
- Rajendra Patil, Harsha Dudeja, Snehal Gawade, et al. “Protocol Specific MultiThreaded Network Intrusion Detection System (PM-NIDS) for DoS/DDoS Attack Detection in Cloud”. In: 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT). ISSN: null. July 2018, pp. 1–7. doi: 10.1109/ICCCNT.2018.8494130.
- Ring, M., Wunderlich, S., Scheuring, D., Landes, D., & Hotho, A. (2019). A survey of network-based intrusion detection data sets. *Computers & Security*, 86, 147-167.
- Roesch, M. (1999, November). Snort: Lightweight intrusion detection for networks. In *Lisa* (Vol. 99, No. 1, pp. 229-238).
- S. KUMAR and K. M. CARLEY 2016, Understanding DDoS cyber-attacks using social media analytics, 2016 IEEE Conference on Intelligence and Security Informatics (ISI), Tucson, AZ, pp. 231–236,
- Sahay, and G. Blanc 2017, “ArOMA: An SDN based autonomic DDoS mitigation framework,” *computers & security*, 70, 482-49
- Santos, R., Souza, D., Santo, W., Ribeiro, A. and Moreno, E. (2019). Machine learning algorithms to detect ddos attacks in sdn, *Concurrency and Computation: Practice and Experience* p. e5402. JCR Impact Factor: 1.167 (2019).
- Sarioguz, O., & Miser, E. (2024). Artificial intelligence and participatory leadership: The role of technological transformation in business management and its impact on employee participation. *International Research Journal of Modernization in Engineering, Technology and Science*, 6(2), Article 1618. <https://www.doi.org/10.56726/IRJMETS49539>
- Schneider, P. (2015). SDN security : Nokia Research perspective . Nokia Solutions and Networks sdxcentral. (n.d.). Why SDN or NFV Now? www.sdxcentral.com: <https://www.sdxcentral.com/sdn/definitions/why-sdn-software-definednetworking-or-nfv-network-functions-virtualization-now/>, Access Date:15.06.2023
- Sharafaldin, I., Lashkari, A. H., Hakak, S., & Ghorbani, A. A. (2019, October). Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. In *2019 International Carnahan Conference on Security Technology (ICCST)* (pp. 1-8). IEEE.
- Shoeb, A., & Chithralekha, T. (2016, March). Resource management of switches and Controller during saturation time to avoid DDoS in SDN. In *2016 IEEE International Conference on Engineering and Technology (ICETECH)* (pp. 152-157). IEEE.
- Simkin, S. (2017). What Is An Intrusion Detection System? <https://www.paloaltonetworks.com>: <https://www.paloaltonetworks.com/cyberpedia/what-is-an-intrusion-detection-system-ids>
- Sinha, S., Sinha, S., & Karkal. (2018). *Beginning Ethical Hacking with Kali Linux*. Apress.
- Tama, B. A., Patil, A. S., & Rhee, K. H. (2017, August). An improved model of anomaly detection using two-level classifier ensemble. In *2017 12th Asia joint conference on information security (AsiaJCIS)* (pp. 1-4). IEEE.
- Tandon, Rajat. (2020). A Survey of Distributed Denial of Service Attacks and Defenses. 10.48550/arXiv.2008.01345. <https://doi.org/10.47672/ajce.2120>
- 79 Forbacha, et al. (2024)

- Tang, T. et al., (2016, October). Deep learning approach for network intrusion detection in software defined networking. In *2016 international conference on wireless networks and mobile communications (WINCOM)* (pp. 258-263). IEEE.
- Tavallae, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009, July). A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications* (pp. 1-6). Ieee.
- Tom Fawcett. "An introduction to ROC analysis". en. In: *Pattern Recognition Letters* 27.8 (June 2020), pp. 861–874. issn: 01678655. doi: 10.1016/j. patrec.2005.10.010. url: <https://linkinghub.elsevier.com/retrieve/pii/S016786550500303X> (visited on 02/26/2023)
- Visual Studio Code—Code Editing. Redefined.* (n.d.). Retrieved May 12, 2023, from <https://code.visualstudio.com/>
- Vinayakumar, R., Soman, K. P., & Poornachandran, P. (2017, September). Applying convolutional neural network for network intrusion detection. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (pp. 1222-1228). IEEE
- Vnware. (s.d.). What is an intrusion prevention system? Accessed at <https://www.vmware.com/topics/glossary/content/intrusion-prevention-system.html>
- Walkowski, D. (2019). What is the CIA Triad. F5 Labs, 9.
- Werlinger, R., Hawkey, K., Muldner, K., Jaferian, P., & Beznosov, K. (2008, July). The challenges of using an intrusion detection system: is it worth the effort?. In *Proceedings of the 4th symposium on Usable privacy and security* (pp. 107-118).
- Yang, C., Liu, J., Kristiani, E., Liu, M., You, I., and Pau, G. (2020). Netflow monitoring and cyberattack detection using deep learning with ceph. *IEEE Access*, 8, 7842-7850. 10.1109/ACCESS.2019.2963716
- Zhang, N., Jaafar, F., & Malik, Y. (2019, June). Low-rate DoS attack detection using PSD based entropy and machine learning. In *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)* (pp. 59-62). IEEE.

License

Copyright (c) 2024 Suh Charles Forbacha, Maah Kelvin Kinteh, Eng. Mohamadou Hamza



This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/).
Authors retain copyright and grant the journal right of first publication with the work simultaneously licensed under a [Creative Commons Attribution \(CC-BY\) 4.0 License](https://creativecommons.org/licenses/by/4.0/) that allows others to share the work with an acknowledgment of the work's authorship and initial publication in this journal.