CHAPTER 1: INTRODUCTION

1.1 BACKGROUND

The setting up and deployment of networks require the expertise of skilled personnel proficient in configuring diverse network components. An extensive approach combining simulation aspects becomes essential in scenarios involving complex interactions between network components such as switches and routers. However, accomplishing this task presents challenges due to the limited programming interfaces available in contemporary networking equipment and the operational costs associated with provisioning and managing extensive multi-vendor networks, encompassing various technologies, which have been steadily increasing. This tendency is accompanied by a decrease in operational revenue and service providers confront additional issues due to a lack of human resources and growing real estate expenses, resulting in a convergence of challenges. As a result, there is growing interest in network unification solutions (Sezer et al. 2013).

The rapid advancement in technological developments has led to significant changes in the utilization of different technologies within enterprise networks. The adoption of system virtualization, cloud computing, fog computing, and mobile networks are clear examples of this change. Consequently, the need for a more adaptable and dynamic network architecture arose to effectively manage the intermediary network devices facilitating these technologies. The idea of a software-defined network (SDN) was developed in response to this demand (Mishra, 2019).

SDN separates the control and data planes, allowing for centralised management and control of network resources via a software-based controller. The OpenFlow protocol, a major component of SDN, allows communication between the controller and the network switches (Anon.2016). The OpenFlow (OF) protocol was one of the first and most widely used techniques to support SDN architecture. OpenFlow allows the control-plane programmability of flow-based switching substrates to be exported. As a result, OpenFlow can be effectively employed in a wide range of applications such as traffic management, load balancing, routing, firewall configuration, and so on that may refer to specific flows that they regulate (McKeown et al. 2008). Software-Defined Networking (SDN) has transformed network management by

providing flexibility and dynamism. This breakthrough, however, has attracted hostile actors looking to exploit flaws in the OpenFlow protocol, which serves as the backbone of SDN. These OpenFlow protocol-based attacks pose serious concerns to network security, integrity, and availability. (Tiwari, et al., 2014).

To protect key infrastructures and maintain the trustworthiness of SDN environments, it is vital to build robust and effective methods for detecting and mitigating such attacks(Khan, et al., 2017). Traditional security solutions may not be adequate in dealing with these threats' growing and sophisticated nature. As a result, there is an urgent need for a sophisticated and proactive methodology capable of detecting and combating OpenFlow protocol-based assaults, hence ensuring the resilience and reliability of SDN systems.

SDN security problems include the possibility of hostile intruders gaining unauthorised access to SDN controllers or switches. Such incursions could result in network reconnaissance, policy reconfiguration, and denial of service attacks, all of which could substantially impact SDN operations and the enterprise networks that rely on them. Extensive research has been devoted to detecting, testing, and fixing security problems in SDN (Ali et al., 2019) (Hameed & Khan, 2018) (Kalkan et al., 2017), however as SDN software evolves, new challenges emerge, needing realistic techniques to test software limitations.

This master's thesis aims to enhance the detection of OpenFlow attacks in Software-Defined Networks (SDNs) by developing an entropy-based method for detecting and mitigating such attacks within the SDN framework. The primary objective is to enhance the security, reliability, and robustness of SDN infrastructures against cyber threats. The thesis will propose viable mitigation strategies to address diverse vulnerabilities, including controller overload, flow table exhaustion, packet-in flooding, and packet buffer overflow. These vulnerabilities are commonly exploited by Distributed Denial of Service (DDoS) and Denial of Service (DoS) attacks. The overall aim of this research is to contribute to the advancement of knowledge in this specialized domain.

1.2 PROBLEM DEFINITION

Software-Defined Networking (SDN) has transformed network management by providing flexibility and dynamism. This breakthrough, however, has attracted hostile actors looking to exploit flaws in the OpenFlow protocol, which serves as the backbone of SDN. These

OpenFlow protocol-based attacks pose serious concerns to network security, integrity, and availability. (Tiwari, et al., 2014).

To protect key infrastructures and maintain the trustworthiness of SDN environments, it is vital to build robust and effective methods for detecting and mitigating such attacks(Khan, et al., 2017). Traditional security solutions may not be adequate in dealing with these threats' growing and sophisticated nature. As a result, there is an urgent need for a sophisticated and proactive methodology capable of detecting and combating OpenFlow protocol-based assaults, hence ensuring the resilience and reliability of SDN systems.

SDN security problems include the possibility of hostile intruders gaining unauthorised access to SDN controllers or switches. Such incursions could result in network reconnaissance, policy reconfiguration, and denial of service attacks, all of which could substantially impact SDN operations and the enterprise networks that rely on them. Extensive research has been devoted to detecting, testing, and fixing security problems in SDN (Ali et al., 2019) (Hameed & Khan, 2018) (Kalkan et al., 2017), however as SDN software evolves, new challenges emerge, needing realistic techniques to test software limitations.

This master's thesis aims to enhance the detection of OpenFlow attacks in Software-Defined Networks (SDNs) by developing an entropy-based method for detecting and mitigating such attacks within the SDN framework. The primary objective is to enhance the security, reliability, and robustness of SDN infrastructures against cyber threats. The thesis will propose viable mitigation strategies to address diverse vulnerabilities, including controller overload, flow table exhaustion, packet-in flooding, and packet buffer overflow. These vulnerabilities are commonly exploited by Distributed Denial of Service (DDoS) and Denial of Service (DoS) attacks. The overall aim of this research is to contribute to the advancement of knowledge in this specialized domain.

1.3 PERSONAL MOTIVATION

As a Cybersecurity student deeply intrigued by the dynamic realm of technology and its worldwide impact, I have a fervent enthusiasm for networking. The numerous network connections and the rise of Software-Defined Networking (SDN) caught my interest in data security and data flow.

My motivation stems from the recognition that SDN is a rising subject within Enterprise networking that requires ongoing inquiry and investigation. I am committed to researching OpenFlow-based threats, methodically studying their patterns, and devising new approaches for their early detection and mitigation. Looking ahead, my goal is to strengthen digital infrastructures and actively contribute to the development of a secure digital landscape.

1.4 RESEARCH QUESTIONS

1. In SDN environments, how can an entropy-based technique identify and counteract emerging OpenFlow attacks?
2. What are the main vulnerabilities in SDN exploited by OpenFlow attacks, and how may they be prioritised for mitigation?
3. How can established security solutions be modified to respond to developing OpenFlow assaults and improve SDN resilience?

1.5 REPORT STRUCTURE

There are six sections in this report: The remaining part of this project is structured into chapters and its contents are presented.

1) Introduction (Chapter One): An overview of the project's structure and content.
2) Literature Review (Chapter Two): A comprehensive examination of existing literature relevant to the project's topic.
3) Project Specifications (Chapter Three): Discussion of project aims, objectives, system requirements, and LESP issues.
4) Design and Justification (Chapter Four): Exploration of design alternatives, justification for the selected design, and machine learning approach. Justification for the chosen ML, SDN controller, and ovs switch for implementation. Comparison of the waterfall approach with an alternative method for detecting OpenFlow protocols.
5) Design, Implementation, and Testing (Chapter Five): Coverage of the Design, Implementation, and Testing phases. Focus on data modelling, execution outcomes, testing on physical/virtual devices, and evaluation procedures.
6) Evaluation of Work (Chapter Six): In-depth explanation of test results, study, and analysis. Comparison of results with literature, project objectives, functional requirements, and contribution to knowledge.

7) Conclusion and Future Work (Chapter Seven): Summary of achieved progress toward project goals. Conclusion of the overall results. Discussion of potential future work and improvements in the field.

CHAPTER 2: LITERATURE REVIEW

This chapter presents a comprehensive literature review focused on this research topic. This chapter aims to explore the existing knowledge, research, and developments in the field, laying the foundation for this study's subsequent analysis and contribution.

2.1 SOFTWARE-DEFINED NETWORKING (SDN)

Software-Defined Networking (SDN) is a game-changing network architecture that overcomes traditional networking constraints. The control interfaces of hosts and switches in traditional networks limit network progress, protocol implementation, and innovation. SDN introduces a centralised software controller, allowing administrators to see and control the whole network. This separation improves flexibility, scalability, and programmability, transforming SDN into a game-changing solution for modern networking difficulties (Raza, 2020; Kaliyamurthy et al., 2021).

SDN is built around a centralized controller that directs network device behavior. This controller serves as the intelligence hub for the network, offering a global view of topology and traffic flow. SDN enables administrators to dynamically administer and reconfigure networks via software policies, easing management and enabling automation (Nunes et al.2014).

2.1.1 Traditional Networking vs. SDN

Traditional networking connects data and control planes firmly, generating network ossification. This ossification impedes the deployment of new apps because they integrate directly into the infrastructure. Furthermore, the variety of control interfaces adds to the difficulty of creating and enforcing regulations. As observed in Content Delivery Networks (CDNs), overlay solutions such as middleboxes (e.g., firewalls, NATs) evolved (Kreutz et al., 2015).

Traditional networks combine control and data operations into equipment such as switches and routers, which complicates network management, especially at scale. This design necessitates manual per-device configuration for policy and protocol modifications, resulting in complex and time-consuming activities.

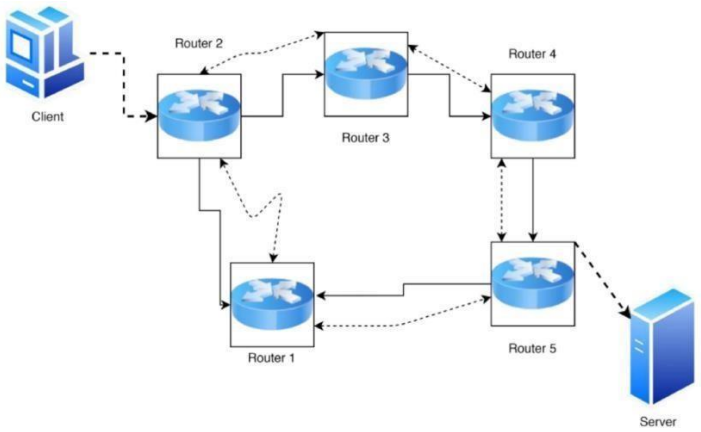


Fig 2.1(A) Traditional Network with distributed control (Satheesh et al. 2020)

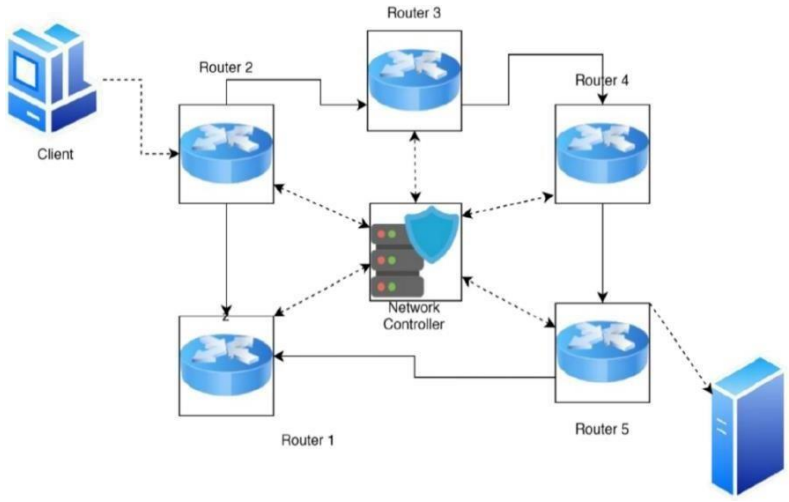


Fig. 2.1(B) Software-Defined Network, with decoupled control (Satheesh et al. 2020).

2.1.2 SDN Architecture and Components

SDN is divided into three layers: infrastructure, control, and application. Understanding SDN's operations and its disruptive networking potential requires an understanding of these layers. Furthermore, significant communication takes place via the Southbound and Northbound interfaces, which are critical for SDN's efficiency and usefulness.

1. Application Layer: The application layer is the highest SDN level, where network management, developers, and users interact. SDN apps use programmability and flexibility to provide a wide range of network services by running atop the control layer. These apps address a variety of requirements, including security, traffic monitoring, analytics, and virtualization. These apps dynamically establish networks, respond to shifts, and handle

application-specific demands by leveraging control layer abstractions. Furthermore, the application layer enables SDN integration with developing technologies such as cloud computing, IoT, and edge computing, opening up new opportunities (Orenda ,2017).

i. Northbound Interface: The interface bridges the control and application levels, allowing SDN apps to communicate with controllers. This gives apps the ability to request resources, access network data, and manage network operations. The Northbound interface serves many SDN app categories such as network monitoring, security, analytics, and virtualization by providing APIs, protocols, and abstractions..

2. Control Layer: The control layer acts as the intelligence hub for SDN, centralising management of the basic infrastructure. The SDN controller serves as the network's decision-maker for behaviour and policy in this layer. The controller communicates with the infrastructure, managing traffic flow, by using a standardised protocol such as OpenFlow.

Dynamic control and programmability rule supreme here. Admins can set policies, traffic regulations, and quality-of-service standards. This layer provides a comprehensive network perspective, allowing for resource allocation, load balancing, and fault management. It also allows for the building and deployment of SDN applications (Alsaeedi et al., 2019).

ii. Eastbound/Westbound Interfaces: Controllers in distributed topologies communicate via eastbound/westbound interfaces. Distributed controllers, for example, connect to share updates and information. Standard eastbound/westbound interfaces are sparse, in contrast to the well-established south/north borders. Each controller has a distinct API, which causes problems in interoperable scenarios (Nkosi et al., 2016).

iii. Southbound Interface: The communication link between the control layer and the infrastructure layer is represented by the Southbound interface. It allows the SDN controller to exchange instructions, directives, and network state information with the forwarding elements in the infrastructure layer. OpenFlow, which provides a standardised mechanism for the controller to connect with network devices, is one of the most extensively used protocols for the Southbound interface. The controller can use the Southbound interface to dynamically configure and reconfigure network devices, govern traffic flows, and enforce network regulations.

3Infrastructure /Data Layer: The infrastructure layer is the heart of SDN, containing physical network devices for data routing. Traditional hardware such as switches, routers, and access points are supplemented with programmable interfaces that are managed by the top SDN layers. This layer coordinates packet mobility in accordance with the SDN controller's instructions (Mckeownick Mckeown, 2008).

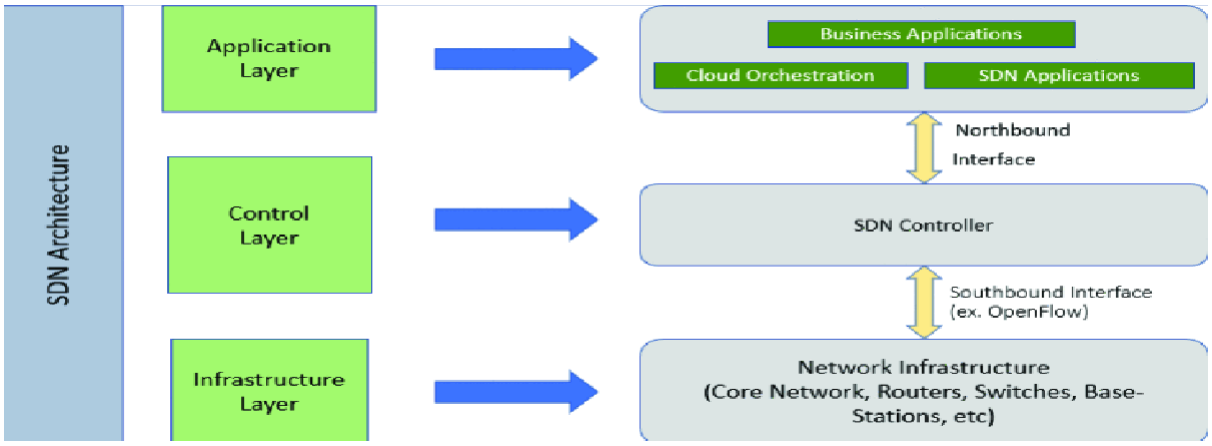


Figure 2.2 Architecture of the SDN components.

2.1.3 Benefits and Challenges of SDN

SDN adds significant benefits and complexities to modern network advancement. SDN provides flexibility and programmability, allowing administrators to regulate network behavior, enforce policies, and quickly reconfigure. This ease facilitates the deployment of new services and apps.

Another advantage is centralized management. A central controller provides administrators with a centralized view of the network, facilitating monitoring, management, and troubleshooting. This improves operational efficiency by reducing the need for manual configuration.

Scalability and efficiency are also noteworthy. Adding and removing devices requires only slight modifications. Adaptive traffic behaviour increases efficiency and optimises resource consumption (Karakus & Durresi, 2017).

However, To understand the security challenges in SDN, we'll use the CIA triad model, which is a paradigm that prioritises data protection within an information system. This notion assists information security experts in customising cybersecurity methods to the needs of various

organisations. The CIA triad is made up of three components: confidentiality, integrity, and availability (DeepWatch, 2020) (ForcePoint, 2021). According to (Priyadarsini & Bera, 2021), SDN security vulnerabilities emerge as malicious attacks on SDN architecture components such as the Control Panel, Data-Plane, Application Plane, and both Southbound and Northbound SDN Interfaces.

Component	Applicable CIA	Prevalent Attacks
Control Plane	Confidentiality	Reconnaissance, snooping, hijacking, SQL injections etc
	Integrity	Tampering, hijacking, replay, spoofing attacks etc
	Availability	DoS, DDoS attacks, hijacking etc
Data Plane	Confidentiality	Reconnaissance, snooping, SQL injections, traditional network attacks etc
	Integrity	Tampering, replay, spoofing attacks etc
	Availability	DoS, DDoS attacks etc
SDN API	Confidentiality	MITM attacks
	Integrity	MITM, repudiation, replay attacks etc
Application Plane	Confidentiality	SQL injection, Tampering, reconnaissance attacks etc

TABLE 1: SECURITY CHALLENGES SUMMARY IN SDN (Odume, O. 2021)

2.2 OPENFLOW PROTOCOL

2.2.1 Overview of OpenFlow

OpenFlow is a Layer 2 protocol that allows SDN components, primarily the controller and switches, to communicate with one another. OpenFlow allows the controller to instruct switches on how to handle incoming packets. These switches function as forwarding devices, following the rules of the controller (Ali et al., 2020). In 2008, Stanford University academics created OpenFlow, which is now managed by the Open Networking Foundation (ONF). Its

widespread industry deployment includes the most recent V1.5 version (Prabha et al., 2022). The controller, OpenFlow switch, and host are typical components of an OpenFlow-based SDN architecture. Notably, the OpenFlow switch plays a critical role in data packet routing and forwarding.

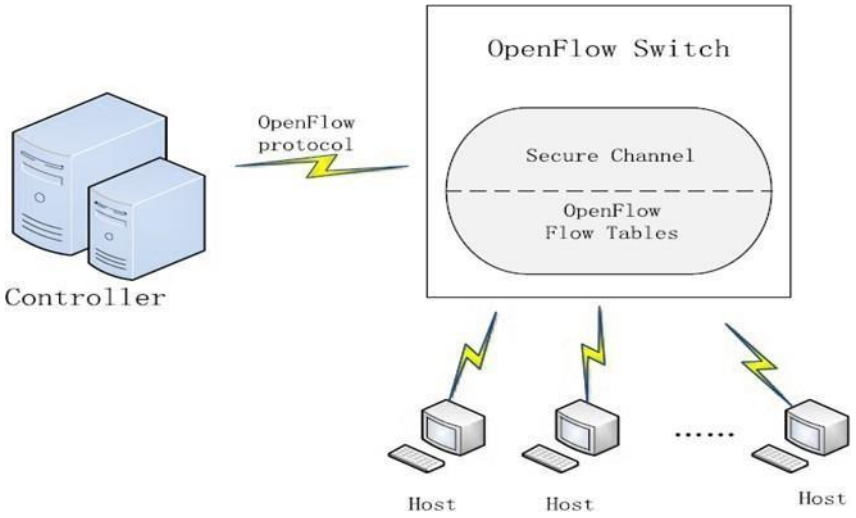


Figure 2.3 OpenFlow-based SDN (Li et al. 2018)

2.2.2 OpenFlow Messages and Flow Table

Messages between the SDN controller and an OpenFlow-based switch are orchestrated by the OpenFlow protocol. This enables controller-led switch programming, which allows for fine-grained control of user traffic flow (Wazirali et al., 2021). The protocol simplifies the creation, modification, and removal of flows, which represent packet sets travelling between network endpoints. Flows provide specific features such as matching attributes, priority, counters, actions, timeouts, and so on. Each flow corresponds to acts guiding switch packet management and is organised in flow tables (Prabha et al., 2022; Wazirali et al., 2021). Key OpenFlow messages include:

a. Features_request: Controller to switch, requesting features.
b. Features_reply: Switch to controller, replying to features_request.
c. Packet_out: Controller to switch, instructing packet actions.
d. Flow modification: Controller to switch, modifying flow table instructions (e.g., add or delete flows).
e. Port modification: Modifying behavior of physical ports.

f. Statistics messages: Includes flow, aggregate flow, table, and port statistics.
g. Packet-In Message: Switch to controller, raised for unmatched packets, seeking controller's actions.
h. Flow Removed Message: Switch to controller, informs about removed flows. Includes details like duration, reason, and counts.

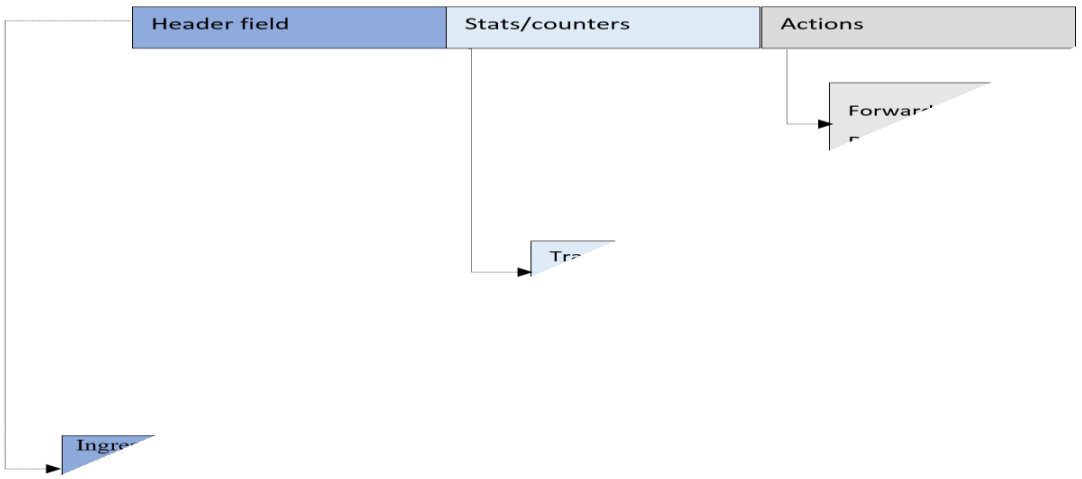


FIGURE 2.4 Flow table entry

The flow table, which is essential in OpenFlow-based switches, contains flow entries that define the criteria for incoming packets and the actions that should be taken. Each entry includes header information, counters, and actions. Headers are used to match arriving packets, counters are used to tally matches, and actions are used to direct treatment (You, Feng, and Sakurai 2017). When packets arrive, they are compared to flow entries. Unmatched packets may be sent to the controller if they are not matched (Prabha et al., 2022; Wazirali et al., 2021). Other flow table entries are dependent on the outcome of the initial match. If the packet matches, the corresponding instructions are executed. Packets can be directed to other tables using the "GotoTable" instruction. Packets without a match and table-miss entries may be dropped.

2.2.3 OpenFlow Versions and Evolution

The OpenFlow protocol has evolved through iterations and enhancements since its initial release in 2009. Versions 1.0, 1.1, 1.2, 1.3, and, most recently, 1.5 have appeared (Huawei., n.d.). Notably, 1.0 and 1.3 were widely used in many networks (Huawei., n.d.; ONF, 2015; James, 2013). The protocol's evolution reflects changing network management requirements and obstacles.

2.2.4 OpenFlow-Based Attacks and Mitigation Strategies

OpenFlow empowers network control but introduces vulnerabilities exploitable by attackers. OpenFlow-based attacks exploit protocol and infrastructure weaknesses, demanding recognition for effective defense. Research addresses OpenFlow-based vulnerabilities and safeguards. Scholars analyze attack vectors, including:

a. Flow Rule Manipulation: Attackers alter flow rules to divert or obstruct traffic, risking interruptions and unauthorized access.
b. Controller Spoofing: Impersonating controllers gains control, causing outages or intrusion.
c. Denial-of-Service (DoS) Attacks: Overwhelming controllers or switches disrupts performance or availability.
d. Flow Table Overflow: Overloading flow tables with entries causes denial of service.
e. Packet Injection: Inserting rogue packets to disrupt traffic or trigger unintended actions.
f. Flow Table Exhaustion: Overwhelming switch flow tables disrupts operation.
g. Controller Overload: Flooding the controller exhausts resources, impacting management.
h. Man-in-the-Middle (MitM): Intercepting communication risks unauthorized access.
i. Data Plane Attacks: Exploiting switch vulnerabilities meddles with traffic.
j. Packet-In Flood: Excessive messages divert controller focus.
k. Data Plane Inconsistency: Mismatched actions disrupt network.

2.2.5 DOS AND DDOS

Given the project's time constraints, our focus will be solely on detecting and mitigating Distributed Denial of Service (DDoS) and Denial of Service (DoS) attacks.

i. Crafted Payload denial of service

This occurs when an attacker creates IP packets that are authentic. The attacker then launches a denial-of-service attack, which can limit resources such as bandwidth, CPU cycles, and memory. IP stack crashing and process table depletion are two examples of DOS. In this case, an assault can be detected by a decrease in service quality.(Gong, 2003).

ii. Volume-based DOS/DDoS

OpenFlow manages incoming data packets based on the matching flow item in the switch's flow table. In the event of a table miss in the flow table, the packet may be routed to the controller for additional processing. This opens the door for attackers to create a Distributed Denial of Service (DDoS) assault. The malicious attacker can construct a large number of packets with missing entries in the flow table and send them to the central controller via OpenFlow, eventually depleting the central controller and failing network performance. Several recent research has found that the SDN paradigm is vulnerable to DDoS attacks by malevolent users(Hameed and Ahmed Khan 2018, Wang et al. 2019).

The following scenarios can emerge from a DDOS attack:

i. Depleting the switch and controller's memory.
ii. Using up all the control channel bandwidth.
iii. Depletion of computing power at the data and control planes.

2.2.6 DoS and DDoS Attack Solutions

The feasibility study by Shin & Gu (2013) highlights that attackers can rapidly generate fake flows, leading to a denial-of-service attack on SDN switches. To counter this, the OpenFlow mechanism initially used a short buffer to drop excessive requests, but it was flawed as it dropped both valid and invalid frames. Wei & Fung (2015) introduced FlowRanger, an algorithm operating on the controller, which prioritizes legitimate data flows during a flooding DoS attack. FlowRanger employs a ranking algorithm to identify regular clients, allowing frames with prior correspondences to pass while denying new corresponding frames.

However, FlowRanger's flaw lies in treating all new corresponding frames as potentially malicious. To address this, a proposed enhancement involves grouping new correspondences using source addresses or header identification. This would allow new single correspondences not in the group to pass.

Another solution, the Trust-Based Attack Detection (TCAD) Model (Priyadarsini, et al., 2019), resembles FlowRanger but employs a signaling game to capture sender behavioral patterns. It uses the STRIDE model and informs the controller to discard and potentially block frames with random behavioral patterns.

Both TCAD and FlowRanger operate solely at the controller layer, similar to the proposed solutions AVANT-GUARD (Shin, et al., 2013) and Topo-Guard (Hong, et al., 2015). The only data plane solution is Sphinx, as suggested by Dhawan et al. (2015). Sphinx observes communication from the controller, identifying relevant OpenFlow messages and comparing flows and policies over time. The various solutions address DoS attacks in SDNs. While FlowRanger and TCAD prioritize legitimate flows, Sphinx operates at the data plane level, distinguishing it from other control plane-focused solutions.

2.3 MACHINE LEARNING

2.3.1 Machine Learning

Machine learning enables computers to take actions without explicit programming (Ng, 2017). SAS defines it as a data analysis technique creating models that learn from data through algorithms, allowing computers to predict accurately (SAS, n.d.). Machine learning includes supervised, unsupervised, and reinforced learning.

Supervised learning predicts labels for unlabeled data using labeled training data, like categorizing 'Anomaly' or 'Normal' traffic. Models are built using this training data and then applied to the test dataset. Algorithm efficiency is assessed by comparing predicted outcomes to actual data (Le, 2016), often evaluated using metrics like precision and recall.

Decision tree algorithms serve as decision support tools, modelling choices and their consequences. They allow researchers to explore options and potential outcomes, aiding in evaluating risks and rewards. Unlike parameterized methods like support vector machines, decision trees are used for classification and regression (Scikit-learn, n.d.).These trees employ entropy and information gain to construct nodes.

Entropy, rooted in Information Theory, measures unpredictability associated with Random Variables or data flow. Entropy values range from [logn, 0], lower when the distribution is constant, and higher when it varies. By comparing packet header field samples' entropy, changes in randomness can be detected.

Unsupervised learning involves analyzing unlabelled datasets to discover relationships between variables. Algorithms infer insights from input data lacking labels (Mathworks, n.d.). An example is clustering.

2.3.2 Existing works of Machine Learning in Network-Related Attacks

Machine learning techniques have been applied to various areas of network-related attacks, including intrusion detection, malware detection, and anomaly detection. Due to evolving network dynamics, intrusion detection systems need to be dynamic and not rely solely on signatures, which can be bypassed by intelligent attackers. This has driven the integration of machine learning into network security, particularly for intrusion detection, to differentiate normal behaviour from anomalies.

Intelligent systems capable of learning new attacks through training with relevant data are required. Various approaches have been put forward, such as SVM combined with dynamically growing self-organizing trees for classification (Khan et al., 2007). Real-time machine learning methods use data features from network packet headers and employ algorithms like decision trees and Naïve Bayes (Sangkatsanee et al., 2011).

However, these machine learning-based intrusion detection systems often produce too many false positives, rendering them less effective (Spathoulas & Katsikas, 2009). To address this, methods like feature selection are proposed to reduce redundant features and enhance classifier accuracy (Anusha & Sathiyamoorthy, 2016; Guo et al., 2010). Feature selection is essential for improving classifier precision, especially in techniques like clustering, which can yield inconsistent results with redundant features (Braga et al., 2010; Tahir et al., 2015). Unsupervised algorithms like Self-Organizing Maps (SOM) also aid in reducing false positives (Alsulaiman et al., 2009).

2.4 RELATED WORK

2.4.1 Existing Security Mechanisms for SDN

Various vulnerabilities and mitigations were covered in a thorough survey on SDN security undertaken by Maleh et al (2022). Security of the crucial control plane in SDN infrastructure is emphasised in the survey. ACLs, firewalls, and IDS are examples of widely used security tools that guard against unauthorised access and harmful activity on the control plane. They might, however, be constrained by dynamic network conditions and find it difficult to respond to sophisticated threats. To overcome the limitations of traditional security mechanisms, researchers have proposed the use of machine learning techniques in SDN security. Machine learning algorithms offer the capability to analyse network traffic patterns, detect anomalies, and identify potential security breaches.

To identify network attacks based on flow-level features, Alzahrani & Alenazi (2021) presented an SDN intrusion detection system employing machine learning techniques (XGBoost, Random Forest, and Decision Tree). In comparison to RF (94.6%) and DT (94.5%), XGBoost received the greatest F1-score (95.55%), and it also exceeded them in terms of precision (XGBoost: 92%, RF: 90%, DT: 90.2%). With a score of 98% for recall, XGBoost showed more stability than RF and DT, which managed 82% and 85%, respectively.

To improve network administration and security, Satheesh et al. (2020) investigate the function of SDN in abnormal intrusion detection and flow-based traffic control. Their suggested priority-based strategy achieves high accuracy rates by effectively utilising bandwidth and utilising machine learning for anomaly identification. It shows higher fault tolerance and quicker routing when compared to traditional methods, demonstrating the promise of SDN for network security and performance improvement. In the suggested priority-based model for anomaly intruder detection and flow-based traffic control, Satheesh et al. (2020) achieved accuracy rates of 79%, 74%, and 82% utilising the data gain, CFS subset, and Gain ratio feature selection strategies, respectively.

Ahuja & Singal (2019) discuss the security issues with SDN, especially DDoS assaults. To identify and stop assaults, they suggest using Open Flow statistics monitoring. The controller reacts if the packet rate goes above a certain point. The study provides information on a few network security strategies, including as mitigation algorithms and entropy-based methods.

The weaknesses of SDN are highlighted by Ali et al. (2020), especially regarding DDoS assaults. They characterise SDN as a paradigm change that enables thorough network programming and centralised control, but they also point out that the OpenFlow protocol contains exploitable flaws. The research focuses on finding infected hosts that produce an excessive number of packets with missing entries, overwhelm the central controller, and degrade network performance. Their suggested method looks at communication patterns to achieve accuracy, efficiency, and resource efficiency. Overall, the study places a focus on fixing SDN vulnerabilities and offers a practical method for identifying compromised servers to lessen DDoS assaults.

Li et al. (2019) propose SA-Detector, a lightweight method for detecting saturation attacks in SDN. They utilize self-similarity characteristics of OpenFlow traffic, measured by Hurst exponents, to identify anomalies. The evaluation in physical and simulation SDN environments demonstrates high accuracy (97.68% and 96.54%) and precision (94.67% and 92.06%) in detecting saturation attacks.

Network security expert Ahalawat et al. (2019) emphasise the growing danger posed by DDoS assaults, particularly UDP flooding attacks. They provide a DDoS detection and rate-limiting mitigation method for SDN that is entropy-based. Improvements in bandwidth utilisation and

hit ratio have been found in tests utilising the Mininet emulator and the Ryu controller, ensuring effective service delivery in the face of DDoS attacks. The study emphasises how SDN's centralised controller makes it susceptible to DDoS attacks.

Moreover, deep learning has shown promising results in enhancing the security of SDN. Deep neural networks can learn complex representations of network traffic and detect sophisticated attacks. For example, Steve (2021) proposed a deep learning-based intrusion detection system for SDN, using different epochs on the BOT-IOT and the IDS-2018 datasets to network to analyse network flow data and identify anomalies. According to the researchers, experimenting with the 50 epochs and 100 epochs, they were able to get the accuracy of 90.77%, 99.95% and 91.21%, 99.95% for the BOT-IOT and IDS-2018 datasets respectively.

Li et al. (2017) investigates the use of deep learning for DDoS attack detection and defence in SDN security. With several configurations, their suggested model, which integrates RNN, LSTM, and CNN, achieves high detection accuracy rates (99.88%, 99.37%, 99.55%, and 99.79%). The model's outputs are used to define drop rules for switches in an OpenFlow-based SDN architecture, which successfully detects and defends against real-time DDoS attacks.

Similarly, the research by Assefa & Oznur (2018), presents a machine-learning framework called MER-SDN. Extraction of features, training, and testing are the three primary phases of machine learning. All investigations were conducted utilising actual network topology and dynamic traffic tracing from SNDlib on Mininet and POX controller. The test results indicate that a 65% reduction in feature size and 70% accuracy have been obtained in the parameter prediction of an energy-efficient exploratory algorithm using the proposed method.

Rasool et al. (2019) explored a deep learning-based approach for traffic classification into malicious and legitimate categories. They simulated a Link Flooding attack that disconnected the controller from the data plane using controlled legitimate traffic. Deep Learning algorithms were applied to classify the traffic post-analysis, utilizing real-time traffic generation through mininet. Artificial Neural Network was used for model training and to predict traffic classes using collected statistics.

Kalkan et al. (2018) introduced a concept combining entropy with TCP layer attributes for attack detection. They emphasized that various attributes like Protocol type, TCP flags, ports, and packet sizes contribute to accurate attack detection. Pairing these features and calculating joint entropy enabled comparison between normal and attacked states. Exceeding the entropy difference threshold led to attack detection, triggering a mitigation phase where traffic ratios were analysed and packets dropped if necessary.

2.5 GAPS OF LITERATURE

The following gaps can be identified:

Underutilization of the "InSDN" dataset: As previously stated, prior research has used datasets such as NSL-KDD, BOT-IOT, and IDS-2018. However, there is a gap in exploiting Elsayed et al.'s (2020) "InSDN" dataset for machine learning based SDN security. This dataset provides current and comprehensive SDN network traffic representation, however, it has yet to be used in SDN Openflow security investigations.

Machine Learning Models with a Limited Spectrum in SDN Security: This section explored machine learning strategies for SDN security, such as XGBoost, Random Forest, and Decision Tree. However, there is still a gap in researching other popular machine learning approaches. Investigating these algorithms may reveal their efficacy in identifying network assaults and anomalies in SDN networks.

A more comprehensive understanding of SDN security mechanisms can be achieved by using the "InSDN" dataset and evaluating additional machine learning techniques. This method would aid in the development of effective intrusion detection and anomaly detection systems within SDN infrastructure.

CHAPTER 3: PROJECT SPECIFICATION

3.1 AIM AND OBJECTIVES

3.1.1 Aim

This project aims to enhance the detection and mitigation of OpenFlow protocol-based attacks in Software-Defined Networking (SDN) environments using machine learning. The project intends to improve the security and resilience of SDN deployments by successfully identifying and mitigating attacks like DDOS and DOS that use OpenFlow protocol vulnerabilities.

3.1.2 Objectives

I. To perform significant research to discover reputable sources for data collection on OpenFlow-based attacks, specifically from SDN networks.
II. Build a robust machine learning model specialized in detecting OpenFlow-based attacks specifically DDOS and DOS, then train the model using the prepared dataset.
III. Rigorously test and validate the trained machine learning model to assess its accuracy and effectiveness in detecting and classifying the chosen attacks.
IV. Create a simulation environment for deploying the trained model on a virtual network device within an SDN environment. Evaluate the model's performance in real-world-like scenarios to demonstrate its efficacy in detecting and mitigating DDOS and DOS attacks in SDN.

3.2 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

3.2.1 Functional Requirements

The functional requirements will be presented using the MOSCOW prioritization approach. The MOSCOW approach involves the characterization of the project requirements based on the level of importance. This method ensures that resources are spent on the most important requirements before the less important ones.

The MOSCOW approach is categorized as Must have, should have, could have and Will not have. This is presented using a tabular approach below

Table 3.1: Functional Requirements Using MOSCOW Approach

S/N	REQUIREMENTS	MOSCOW
1	A total of three (3) machine models will be trained and the one with the best accuracy score will be selected	Must
2	An SDN environment will be created for the deployment of the model to a virtual network device.	Must
3	Large amounts of data will be analyzed to identify patterns and make predictions.	Should
4	The predicted classes will be evaluated using the confusion matrix and accuracy.	Should
5	The model accuracy must be well over 90% in the detection of attack traffics.	Must

6	Integrating with visualization and reporting tools.	Could
7	Deployment of solution as a stand-alone desktop application.	Won't
8	Use built API to deploy the built machine learning model into production.	Could
9	Using the Security information and event management (SIEM) systems for the implementation of machine learning solutions.	Won't
10	The use of Recall, Precision and F-Measure will be used to evaluate predicted classes.	Should
11	The solution will have the ability to automatically perform data processing and feature selection.	Won't
12	The ability to handle incomplete data will be included in the solution.	Won't

3.2.2 Non-functional Requirements

These specifications consider the system's functionality as well as any potential performance limitations.

S/N	REQUIREMENTS	MOSCOW
1	**Performance** The system must be able to quickly and accurately identify and stop OpenFlow-based assaults	Would
2	**Scalability:** The solution ought to be scalable to handle substantial SDN environments with a significant amount of network devices	Should
3	**Usability:** Network administrators must be able to monitor and control threat detection and mitigation with an easy-to-use user interface.	Would
4	**Reliability:** The solution should have a high level of reliability in detecting and mitigating attacks without false positives/negatives	Could
5	**Interoperability:** The system may be able to integrate with different SDN controllers and hardware suppliers for networks.	Could

3.3 SYSTEM REQUIREMENTS

The following are the specifications for the system and applications utilized in the analysis and detection:

1. Operating System: Windows 11 64bit
2. Processor: 2.5 GHz Dual-Core Intel Core i5
3. Storage: 500GB SSD

4. Memory: 16 GB 1333 MHz DDR3
5. Graphic card: Intel HD Graphics 4000 1536 MB
6. Jupyter Notebook 6.3.0
7. Python 3.11.3
8. Mininet
9. Virtual Box 7.0.8
10. Ubuntu 22.04.2
11. Pycharm
12. Open Daylight

3.4 PROJECT METHODOLOGY

The Project methodology includes a well-defined Waterfall model, which consists of successive stages that guide the project's progression from start to finish. Winston W. Royce first defined this method in 1970 ("The Waterfall Development Methodology", 2006). It immediately received management support because everything flows logically from the beginning to the completion of a project (Jonasson, 2008).

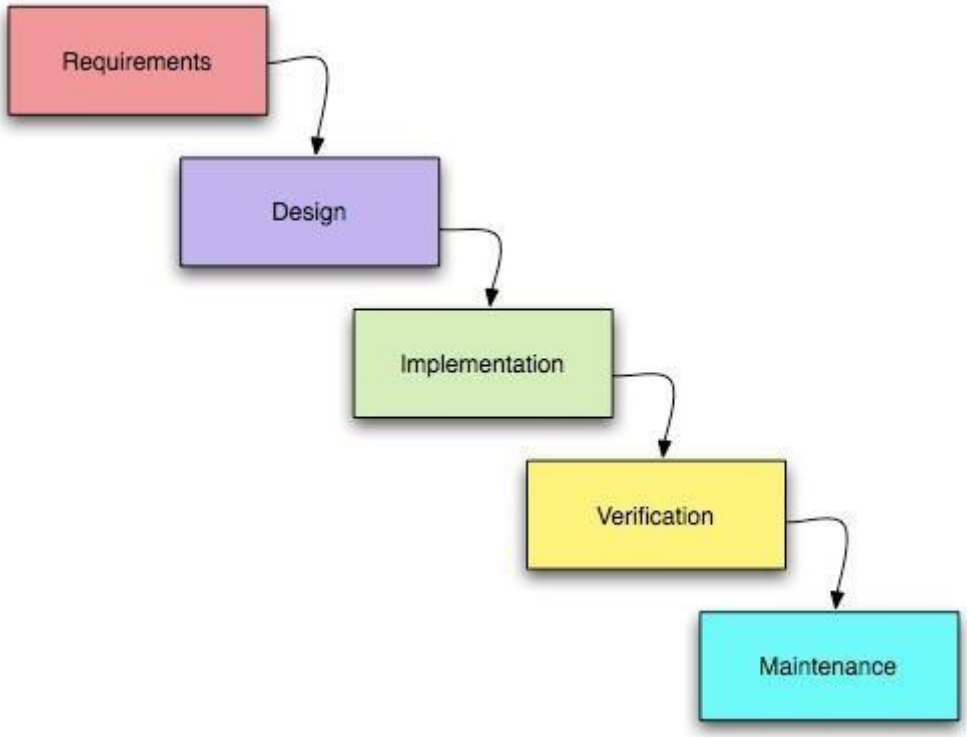


FIGURE 3.1A WATERFALL MODEL

3.5 REVIEW OF LEGAL, ETHICAL, SOCIAL AND PROFESSIONAL ISSUES

With the increasing capability and prevalence of machine learning in cybersecurity, new ways to detect and mitigate OpenFlow protocol-based threats in SDN systems have emerged. As machine learning models improve in sophistication, it is critical to investigate and mitigate potential legal, ethical, social, and environmental issues. We hope to reduce data privacy and intellectual property concerns by using a publicly available dataset.:

1. Legal Issues:

Privacy: Machine learning often involves the collection and processing of large amounts of personal data, raising concerns about privacy and data protection. Regulations such as the General Data Protection Regulation (GDPR) aim to address these issues. When working with sensitive cybersecurity information, data privacy is a top priority.

Intellectual Property: Machine learning models can be based on proprietary algorithms and datasets, leading to intellectual property disputes and challenges around algorithmic transparency.

2. Ethical Issues:

Bias in machine learning models creates ethical quandaries. We may, however, eliminate biases and encourage fairness in the model's conclusions by using a broad and representative publicly available dataset. Open datasets also improve model openness and transparency, ensuring ethical accountability.

3. Social Issues:

Job displacement because of automation is becoming a growing concern. However, by increasing cybersecurity with machine learning models, we want to favorably impact society by protecting key infrastructure from cyber assaults. The use of a publicly available dataset supports knowledge exchange and inclusion while reducing social inequities in technological access.

4. Professional Issues:

Professionals in the fields of cybersecurity and machine learning research must respect ethical standards and prioritise data privacy. Transparency, accountability, and

stakeholder collaboration are critical for building trust in the research and its potential applications.

In addition, the project would follow the IEEE P2840 standard, which ensures that machine learning models developed with encrypted data meet the appropriate security and privacy standards. It provides a structured approach and rules for designing, deploying, and sharing machine learning models safely and privately.

There are various advantages to using a publicly available dataset in the creation and implementation of machine learning models for entropy-based detection and mitigation of OpenFlow protocol-based attacks in SDN. This research intends to positively contribute to cybersecurity while assuring responsible and sustainable technical growth by proactively addressing legal, ethical, social, and professional problems.

CHAPTER 4: DESIGN AND JUSTIFICATION

This chapter discusses the design phase for developing a model and system architecture for successful project implementation and evaluation. It also covers the justifications for the design, software, and programming language used.

4.1. SYSTEM DESIGN

Our system design method is inspired by Whitten and Bentley (2006), who states that a comprehensive design includes architecture, components, modules, interfaces, and data. Furthermore, Bass, Clements, and Kazman's (2003) emphasis on the critical role of system design in software development resonates well with our research goals.

4.1.1 System Architecture

Our system architecture is the basic blueprint that controls the orchestration of components and their interactions. It serves as the foundation for the entropy-based detection and mitigation system. The system architecture for these projects is divided into two sections.

- Machine Learning development approach
- SDN testbed network architecture

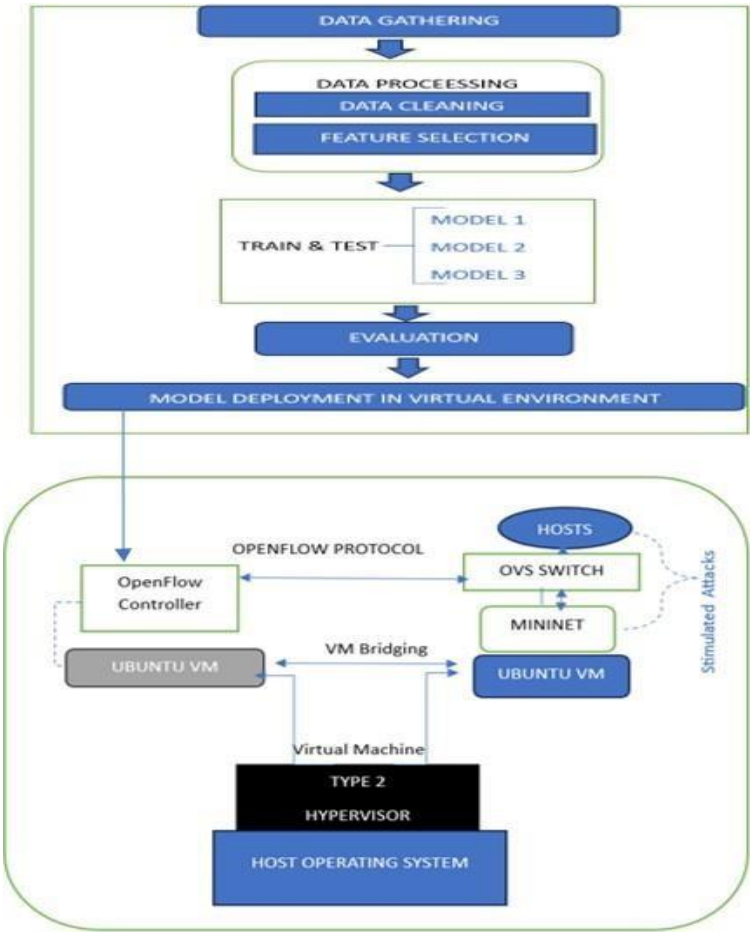


FIGURE 4.1: System Architecture

4.2 SDN testbed and Alternatives

The virtual SDN testbed network architecture provides a controlled environment for building and assessing an entropy-based detection and mitigation system for OpenFlow-based threats in an SDN context. This architecture simulates real-world network scenarios, allowing for rigorous system testing and validation. Various components have been utilized for the design and execution of this project which are:

1) Operating Systems

The chosen operating systems are Linux-based. They will be implemented within a virtualized computing environment, hosted on a Windows OS.

2) Network Emulator

For the emulation of network scenarios, the Mininet stable release 2.3.0 (GitHub, 2021) will be employed. This widely recognized tool is commonly utilized for network research purposes. It encompasses multiple bug fixes and incorporates support for Python, a crucial element for this specific test case. Mininet facilitates the creation of practical virtual networks aligned with the test case's objectives. This empowers users to establish diverse topologies within SDN networks, as well as to analyze, modify, and collaborate on various network components.

To install Mininet NS, the instructions provided here (GitHub, 2021) should be followed. It is recommended to utilize a combined Ubuntu and Mininet image.

Advantages of Mininet

a) It starts networks quickly; it only takes a few seconds.

b) Creating customised topologies, from a single switch to the backbone or data centre topologies, is simple.

c)It is open source, can run on a laptop like this one, and supports customised packet forwarding.

Mininet constraints

a) Since it virtualizes hosts using a single Linux kernel, you cannot run software that depends on other operating system kernels.

b) Because Mininet hosts share host file systems and process ID spaces, executing daemons that need to be configured in /etc requires extra caution.

c) The researcher, rather than Mininet, configures and programmes the controller.

3)SDN Controller

This project would be using the Ryu controller for SDN stimulation (Ahuja and Singal Dec 2019). A framework for creating and implementing software-defined networking (SDN) applications is offered by the robust and adaptable open-source RYU controller. Its wide range of features, advantages over other programming languages like Java, and use of Python makes

it a preferred option for researchers, developers, and network administrators, which is why it was picked. Many controllers are accessible under open-source licences, making them freely useable. Although there is excellent support for using controllers like Floodlight and open daylight, their learning curve is somewhat high, especially given that Java is their development language. Due to its very flat learning curve, the RYU controller is used in this research over other controllers.

4.2 JUSTIFICATION OF DATASET

For this project, the InSDN dataset was used in training the machine-learning model to help in detection. The Attribution-NonCommercial 4.0 International (CC BY-NC 4.0) licence applies to this InSDN dataset. InSDN is a comprehensive Software-Defined Network (SDN) dataset for Intrusion detection system evaluation. The new dataset comprises several attack categories that can happen in different SDN standard components, both benign and malicious. DoS, DDoS, brute force attacks, web applications, exploitation, probes, and botnets are just a few of the attacks that InSDN takes into account. Additionally, the created data's typical flow includes a variety of widely used application services, including HTTPS, HTTP, SSL, DNS, Email, FTP, and SSH(Elsayed, Le-Khac and Jurcut 2020). The lack of easily available datasets suited for direct application in anomaly detection systems within SDN networks was the driving factor behind the choice of InSDN for this project. The KDD'99 dataset, NSL-KDD, Kyoto dataset 2006, ISCX2012, CICIDS 2017, and CSE-CIC-IDS2018 are only a few instances of the many current research activities that rely on out-of-date and incompatible datasets.

4.3 JUSTIFICATION OF METHODOLOGY

The waterfall approach is a traditional linear and sequential software development methodology used for managing projects with stable, well-defined requirements. It follows a clear sequence where each phase must be completed before moving on to the next, akin to water flowing over a waterfall. This approach is suitable for this project for the following reasons:

1 Well-Defined Stages: The Waterfall approach offers precisely delineated stages, allowing efficient project management and tracking. For instance, in developing an intrusion detection system, stages may include issue description, data gathering, data pre-processing, model selection, training, testing, deployment, and maintenance.

2 Fixed Requirements: With fixed requirements, success can be assessed based on the system's ability to effectively detect intrusions. The Waterfall methodology ensures an early definition of criteria and their fulfilment throughout the project's lifecycle.

3 Traceability: The Waterfall approach facilitates documentation and tracking of requirements, design, and testing, crucial for auditing and regulatory compliance in intrusion detection systems.

4 Proactive Issue Resolution: The sequential nature of the Waterfall approach allows for the early identification and resolution of issues at each stage. This proactive approach minimizes the risk of significant problems arising late in the project.

Effective Resource Management: By establishing a fixed budget and timeline at the outset, the Waterfall method enables efficient resource management, ensuring timely completion within defined constraints.

CHAPTER 5: IMPLEMENTATION AND TESTING

5.1 INTRODUCTION

This chapter focuses on simulating and evaluating the proposed Entropy-Based Detection and Mitigation system for OpenFlow Protocol-Based Attacks in Software-Defined Networking. It covers data gathering, creating machine learning models, and putting them into practice virtually using Mininet and RYU Controller. Simulations involving different assault scenarios are used to properly assess the system's precision, effectiveness, and performance.

5.2 MACHINE LEARNING WORKFLOW

The machine learning workflow is a structured and systematic approach to developing and evaluating machine learning models (Ng, Jiang and Chow 2020).

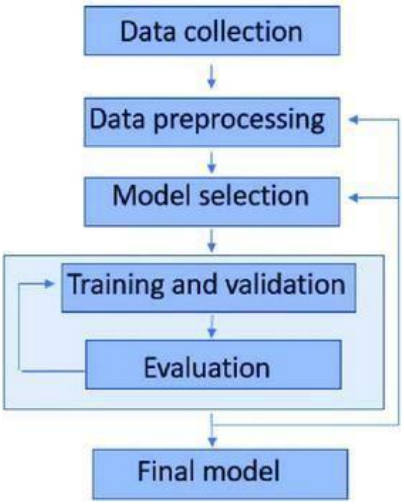


FIGURE 5.1: MACHINE LEARNING WORKFLOW DIAGRAM

5.2.1 Dataset Collection and Entropy

In machine learning, Entropy is a measure of the impurity or uncertainty of a dataset. Entropy-based detection is employed to identify OpenFlow attacks by examining the entropy of particular fields or characteristics in the OpenFlow messages that are transmitted between the controller and the switches. Entropy, which gauges data's randomness or predictability, comes into play. If there are anomalies in the OpenFlow protocol, such as unusual patterns or values in the message fields, the entropy level may rise as a consequence.

For this research, the dataset utilized was gathered from InSDN, a comprehensive Software-Defined Network (SDN) dataset tailored for evaluating intrusion detection systems. Mahmoud Said Elsayed, Nhien-An Le-Khac, and Anca Jurcut compiled this dataset, which was published in the "InSDN: SDN Intrusion Dataset" (submitted to IEEE Access Journal, 2020). The statistical measures were used in capturing the entropy of the features.

The dataset includes both benign and diversified attack categories, as well as different SDN standard aspects. It includes a wide variety of attacks, including DoS, DDoS, brute force, web applications, exploitation, probing, and botnet attacks. Our selection of this dataset stems from its robust nature and congruence with our research objectives. The dataset aggregates

343,939 instances in entirety, comprising both normal and attack-driven traffic. Normative instances tally 68,424, while attack-driven instances total 275,515. This comprehensive dataset affords a multitude of avenues for efficient machine learning model training and evaluation.

For project scope, we narrow our focus exclusively to DDoS and DoS attack classifications. Consequently, the dataset under consideration features 73,529 DDoS attacks, 53,616 DoS attacks, and 68,424 instances representative of normal network traffic.

5.2.2 Data Cleaning

Data collecting frequently confront challenges such as inconsistencies and inaccuracies, which might jeopardize the data's dependability and quality. Poor performance might be caused by dirty data in the machine-learning model. Missing values, duplicate records, inconsistent formatting, outliers, inaccuracies, contradicting information, and data integrity issues are all signs of dirty data.

Data cleansing is important in ensuring the data's appropriateness for machine learning and analysis. This procedure entails dealing with missing numbers, removing duplicates, standardizing formats, discovering and resolving outliers, validating data, and deleting inconsistent or inaccurate values. These actions considerably increase the data's dependability, precision, and usefulness for analysis. The Python programming language will be used to efficiently carry out the data purification operation. The following checks were carried out:

a. Missing Values Check

```
In [10]: # Check for NaN values in the DataFrame
         nan_present = df_c.isnull().any().any()
         nan_present

Out[10]: False
```

b. Duplicate Values

```
In [12]: duplicates = df_c.duplicated().all()
         duplicates

Out[12]: False
```

The dataset was confirmed to be clean and all that was left to do was to perform feature selection on it which would be used to train the machine model.

5.2.3 Feature Selection

Although the dataset contains 84 features, not all of them are important for identifying malware traffic over HTTPS connections. The random forest classifier uses the average

decrease impurity approach to locate the critical characteristics. Based on the label distribution, this technique determines how frequently a selected piece from the dataset would be mislabelled. The model's performance in detecting malware traffic can be improved by focusing on the most significant features.

The formula for calculating the average decrease impurity of a feature can be expressed as:

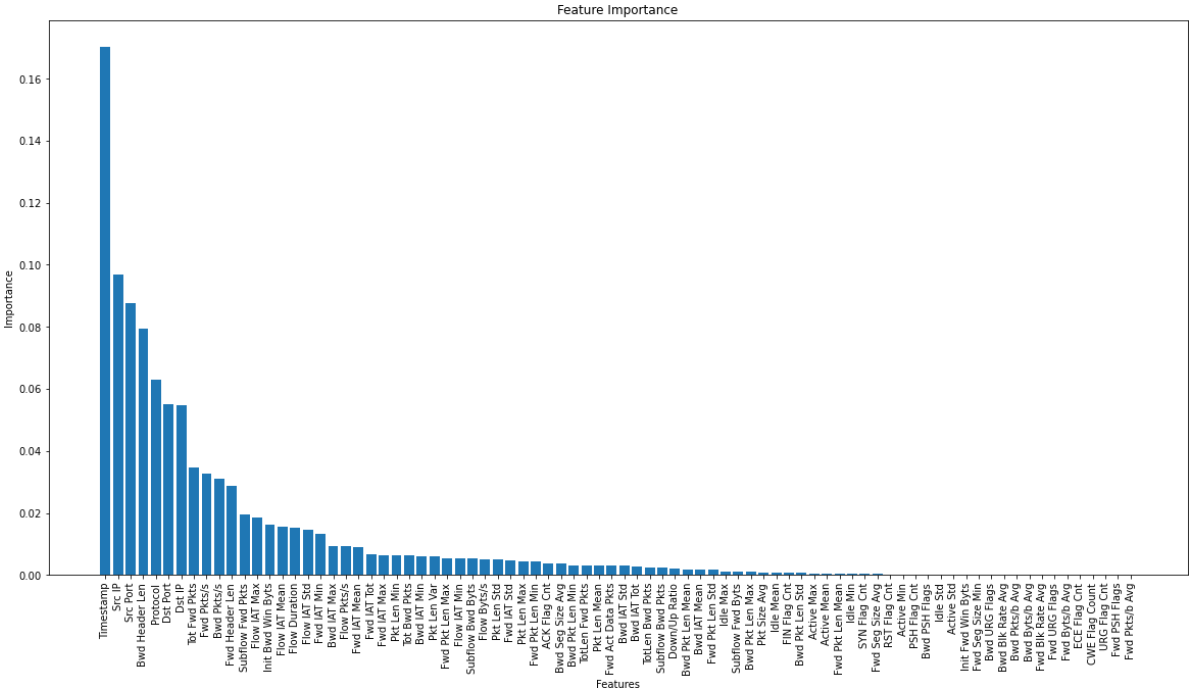
Average decrease impurity = sum ((p * (1 - p) * w) / (total impurity)),

where:

- p is the probability of a sample being classified to a specific class within a node,
- w is the number of samples in the node,
- total impurity is the overall impurity measure in the node.

The feature importance is then normalized by dividing the Average decrease impurity values by the sum of all feature importance values so that they add up to 1.

Figure 5.2 shows the graph of feature importance using the Random Forest classifier.



From the above graph, as it can be seen, a good portion of the features have little or no importance to achieving the goal of being able to detect the presence of malicious traffic in HTTPS connections. The features with higher importance will be selected in the building of the malware detection model.

Fig.5.3 shows the feature importance heatmap of a sample of the dataset.

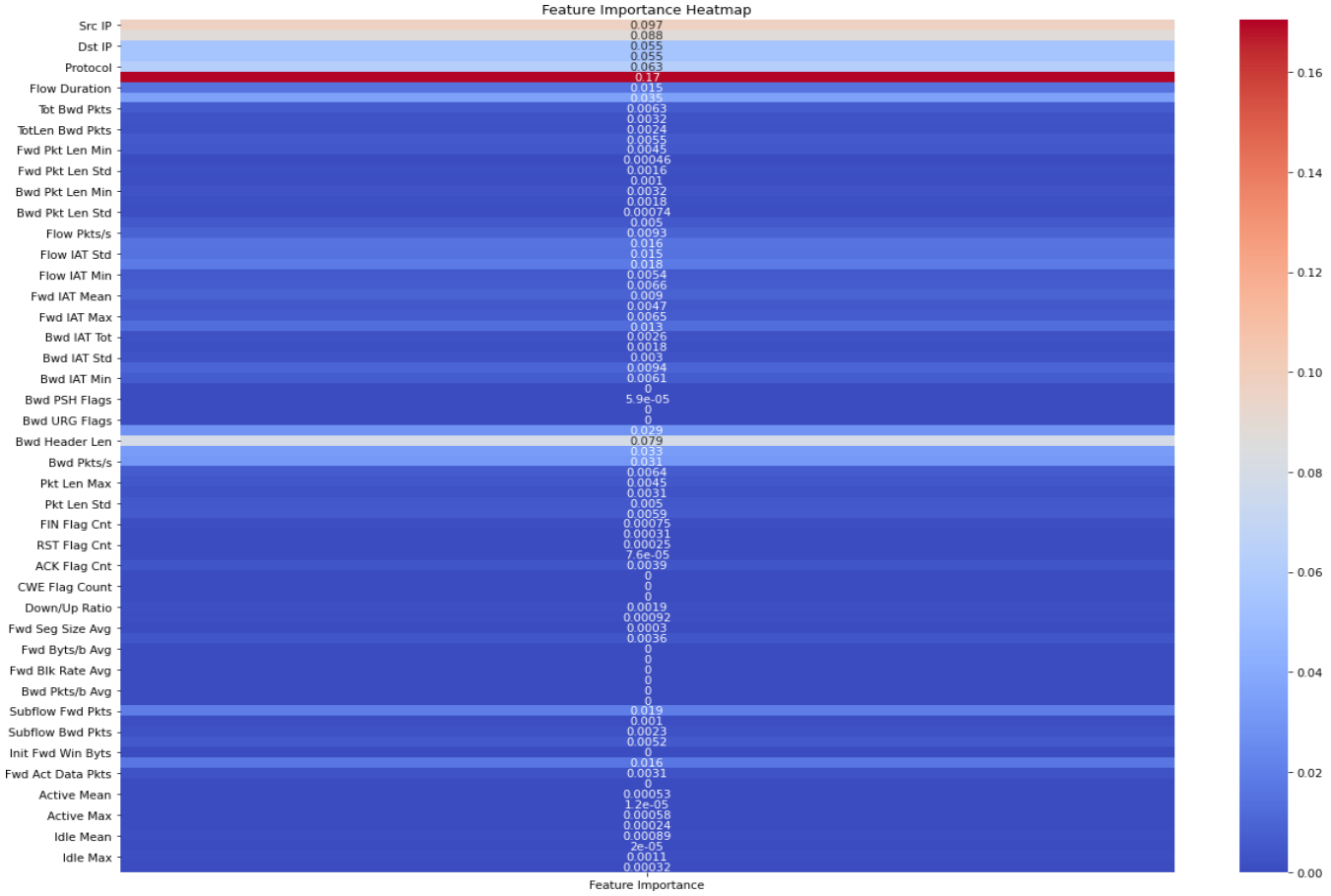


Fig. 5.3. Feature Importance Heatmap

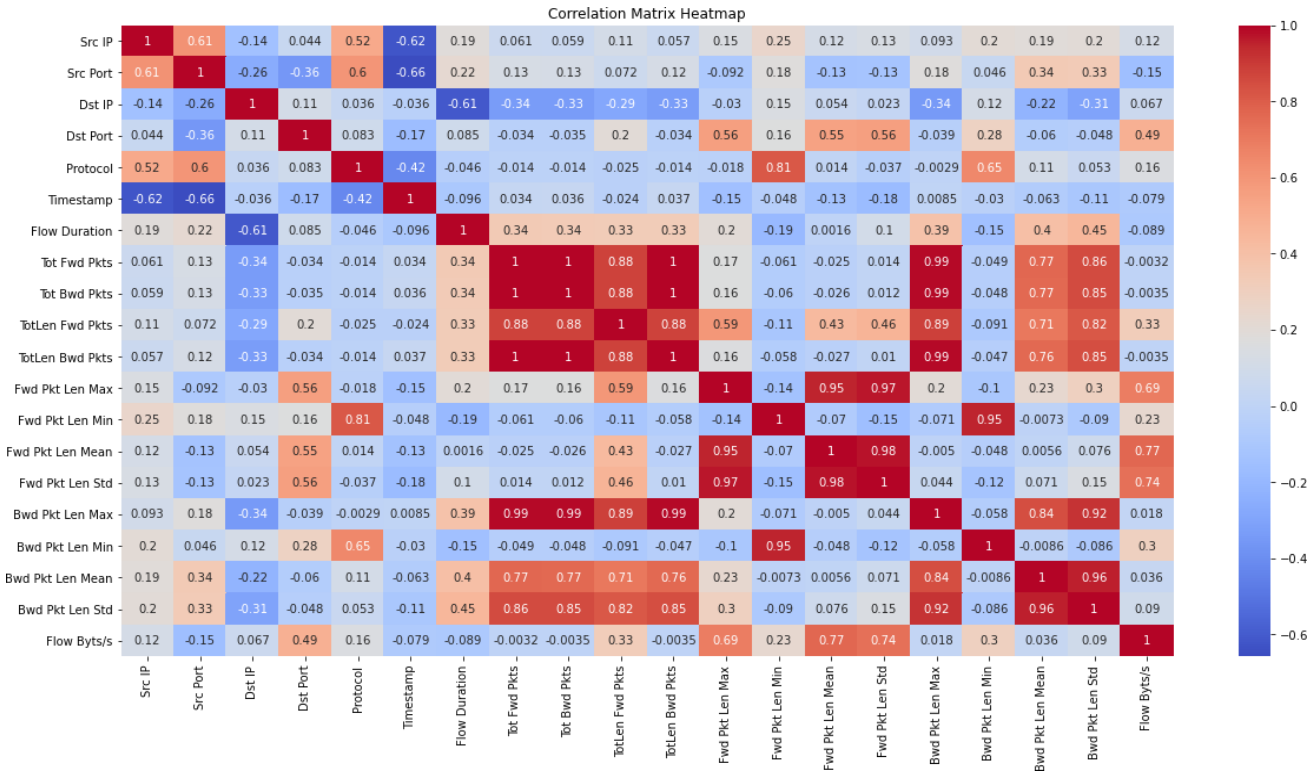


Fig.5.4. Correlation Matrix Heatmap

```
In [26]: # Load your dataset and split it into features (X) and target variable (y)
         X = df_rand.drop('Label' , axis = 1)
         y = df_rand['Label']
         # Create a Random Forest model
         rf = RandomForestClassifier()
         # Perform feature selection
         selector = SelectFromModel(estimator=rf)
         selector.fit(X, y)
         # Obtain the selected features
         selected_features = X.columns[selector.get_support()]
         print(selected_features)

Index(['Src IP', 'Src Port', 'Dst IP', 'Dst Port', 'Protocol', 'Timestamp',
       'Flow Duration', 'Tot Fwd Pkts', 'Flow Byts/s', 'Flow Pkts/s',
       'Flow IAT Std', 'Flow IAT Max', 'Fwd IAT Min', 'Fwd Header Len',
       'Bwd Header Len', 'Fwd Pkts/s', 'Bwd Pkts/s', 'Subflow Fwd Pkts',
       'Init Bwd Win Byts'],
      dtype='object')

In [27]: len(set(selected_features))

Out[27]: 19
```

Fig.5.5. Selected Features for machine learning building

The entropy-based collected features were selected using the random forest classifier.

```
In [28]: # Fit the classifier to the data
         rf.fit(X, y)

         # Get the feature importance scores
         feature_importances = rf.feature_importances_

         # Create a DataFrame with feature names and importance scores
         feature_importances_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})

         # Sort the DataFrame by importance scores in descending order
         feature_importances_df = feature_importances_df.sort_values('Importance', ascending=False)

         # Print the feature importance scores
         print(feature_importances_df)
```

```
           Feature  Importance
5        Timestamp    0.170390
0           Src IP    0.096931
1         Src Port    0.087770
40  Bwd Header Len    0.079418
4         Protocol    0.062826
..             ...         ...
55    ECE Flag Cnt    0.000000
54  CWE Flag Count    0.000000
53    URG Flag Cnt    0.000000
35   Fwd PSH Flags    0.000000
61  Fwd Pkts/b Avg    0.000000

[82 rows x 2 columns]
```

```
In [29]: final_feat = feature_importances_df.loc[feature_importances_df['Importance'] >= 0.02, 'Feature'].tolist()

In [30]: final_feat.append('Label')

In [31]: final_feat

Out[31]: ['Timestamp',
          'Src IP',
          'Src Port',
          'Bwd Header Len',
          'Protocol',
          'Dst Port',
          'Dst IP',
          'Tot Fwd Pkts',
          'Fwd Pkts/s',
          'Bwd Pkts/s',
          'Fwd Header Len',
          'Label']
```

Fig.5.6. Final Selected Features with feature importance

The final features that were used in building the machine-learning model cab seen in Fig. 5.6.

5.2.4 Data Set Splitting

As we have an explicitly labelled target variable, we will use supervised learning to develop a machine model capable of detecting DDoS and DoS traffic in this project.

The data set will be divided into two halves with an 80:20 proportion. 80% of the data, which includes both regular and malicious traffic, will be used to train the machine model. The greater portion is critical for giving enough data to adequately train the model. The remaining

20% will be used as testing data, allowing us to assess how well the machine model can distinguish between normal and malicious traffic.

The machine learning model's accuracy can be calculated using the following formula: Accuracy = Number of Correct Predictions / Total Number of Predictions.

5.2.5 Model Development

We are now ready to train the model to reliably detect both types of traffic connections after completing all of the necessary phases in the machine learning process, including the correct labelling of malicious and normal data. Because classification was chosen as the method for training the model, the machine learning model is a classifier. Given that our dataset contains well-defined target labels, supervised learning is the best option.

We will create and test three likely models, which will be chosen based on their popularity and performance on datasets with comparable data structures to our project. The model with the best degree of accuracy in distinguishing between regular and malicious traffic will be chosen as the final option.

5.2.6 Testing Model

After completing the successful training of my machine model, the next step is to test how it accurately predicts the target variable. The models built will be tested on the reserved 20% of the dataset, this refers to the traffic the models have not seen before and it will be used to test how efficient the machine model is at classifying both the malicious and normal traffic.

Three machine models were trained; therefore, three machine models will be tested for predictive accuracy.

The attacks that will be detected by the trained machine learning model will be DDoS, and DoS which will be represented by 1, 2 respectively and the normal traffic will be represented by 0.

1. **Random Forest Model**

The random forest model produced an accuracy of 100% after testing it on the 20% reserved data that the model had not seen.

Fig.5.7. shows the confusion matrix of the Random Forest Model. The confusion matrix compares the predicted class with the true class and as can be seen from the diagram, the model performed well in classifying both the normal and malicious data.

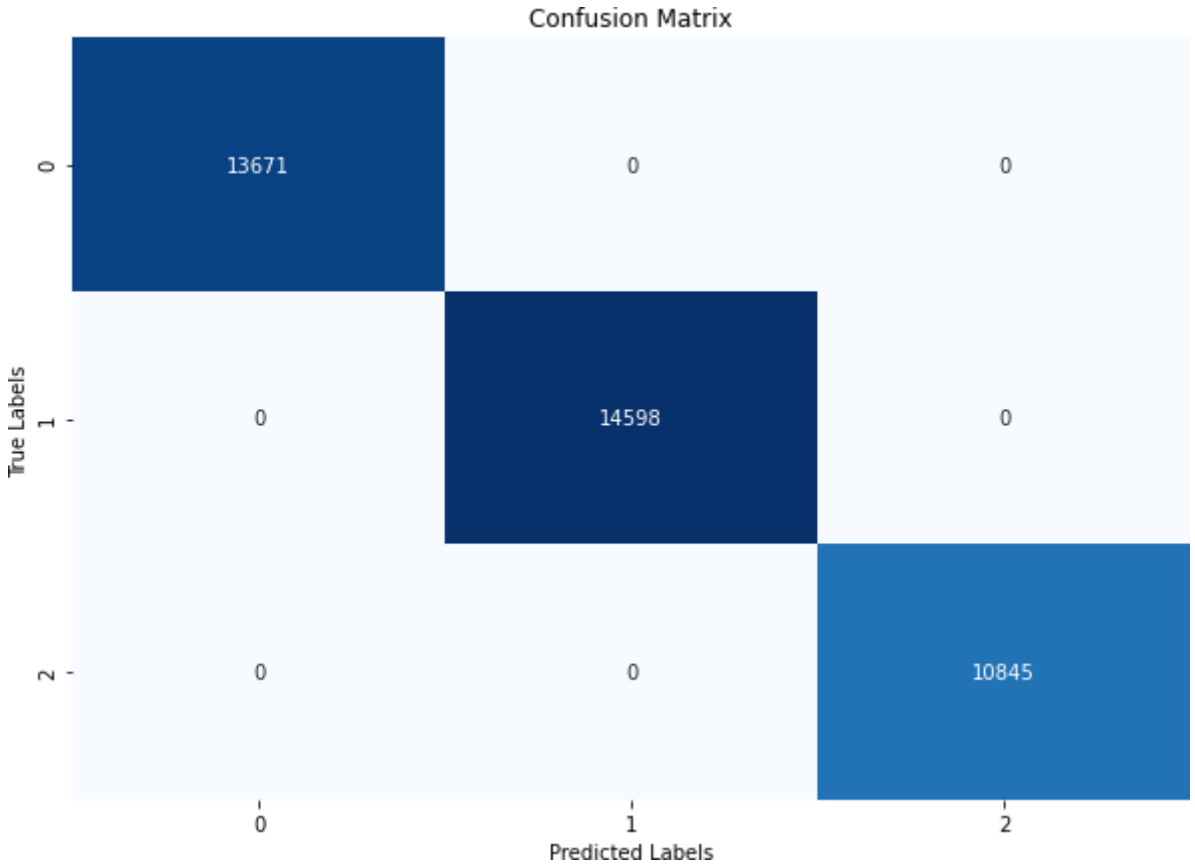


FIGURE 5.7 Confusion Matrix Heatmap

```
In [45]: import numpy as np
         from sklearn.metrics import precision_score, recall_score, f1_score

         # predicted classes and true labels stored in 'y_pred_rf' and 'y_test' arrays respectively

         # Calculate Precision
         precision = precision_score(y_test, y_pred_rf, average='weighted')

         # Calculate Recall
         recall = recall_score(y_test, y_pred_rf, average='weighted')

         # Calculate F1-Score
         f1 = f1_score(y_test, y_pred_rf, average='weighted')

         print("Precision:", precision)
         print("Recall:", recall)
         print("F1-Score:", f1)

         Precision: 1.0
         Recall: 1.0
         F1-Score: 1.0
```

Fig.5.8. Precision, Recall and F-Measure

2. K-Nearest Neighbor Model

The KNN model after training it with 20% of the reserved data was able to classify with 99.9% accuracy both the normal and malware traffic.

Fig. 5.9 shows the confusion matrix of the KNN model

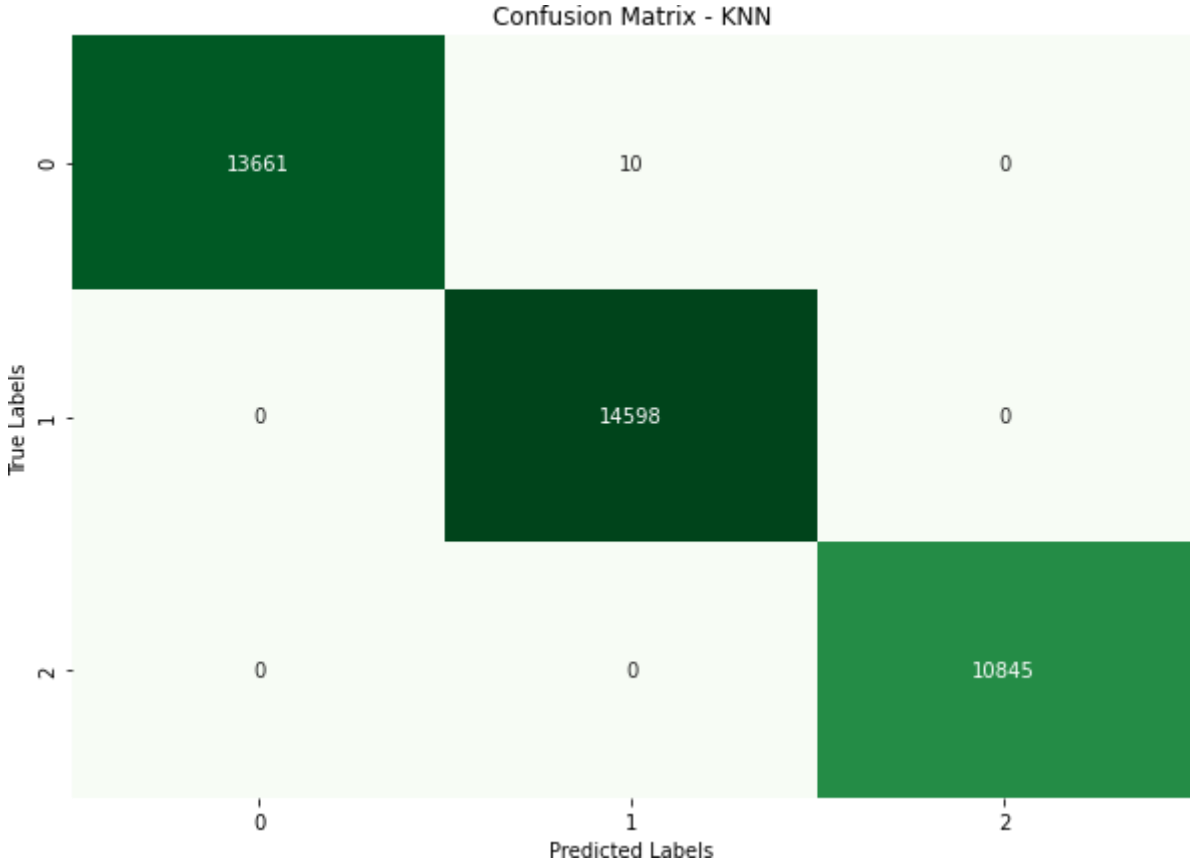


FIGURE 5.9 KNN Confusion Matrix Heatmap

The confusion matrix was able to compare the predicted class with the true class and as can be seen from the diagram, the model performed well in classifying both the normal and malicious class

```
In [54]: from sklearn.metrics import classification_report

         # Make predictions on the test set
         y_pred = knn.predict(X_test)

         # Calculate Precision, Recall, and F1-Score
         report = classification_report(y_test, y_pred)

         print("Classification Report:")
         print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     13671
         1.0       1.00      1.00      1.00     14598
         2.0       1.00      1.00      1.00     10845

    accuracy                           1.00     39114
   macro avg       1.00      1.00      1.00     39114
weighted avg       1.00      1.00      1.00     39114
```

Fig.5.8. Precision, Recall and F-Measure

3. Naive Bayes Model

The Naïve Bayes model after training it with 20% of the reserved data was able to classify with 99.9% accuracy both the normal and malware traffic.

Fig. 5.10 shows the confusion matrix of the model.

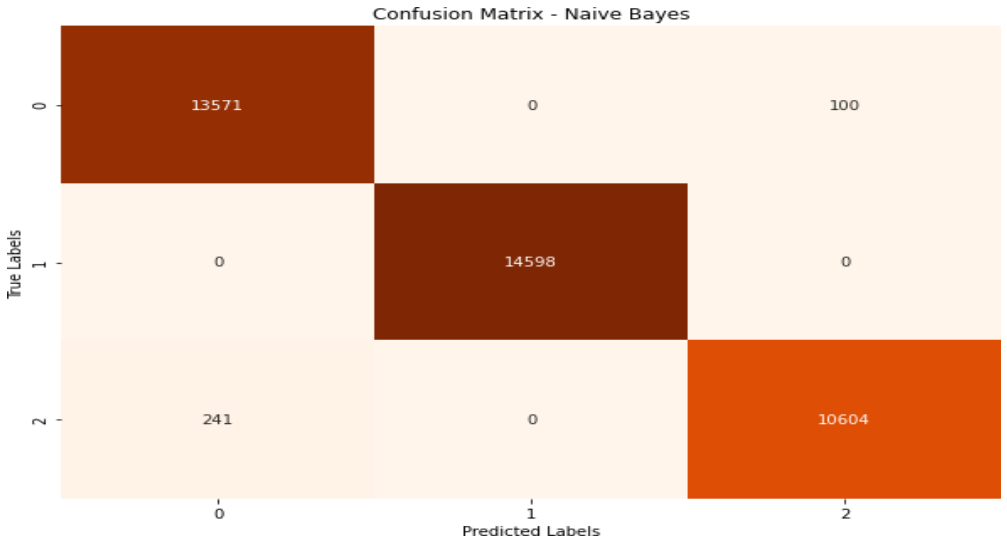


Fig. 5.10 confusion matrix of the model.

```
In [58]: # Make predictions on the test set
         y_pred = naive_bayes_classifier.predict(X_test)

         # Calculate Precision, Recall, and F1-Score
         report = classification_report(y_test, y_pred)

         print("Classification Report:")
         print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.99     13671
         1.0       1.00      1.00      1.00     14598
         2.0       0.99      0.98      0.98     10845

    accuracy                           0.99     39114
   macro avg       0.99      0.99      0.99     39114
weighted avg       0.99      0.99      0.99     39114
```

Fig.5.10. Precision, Recall and F-Measure

Naïve Bayes Heatmap

The confusion matrix was able to compare the predicted class with the true class and as can be seen from the diagram, the model performed greatly in classifying both the normal and malicious class.

5.3 Virtual Implementation

The machine model has been trained successfully and the performance was outstanding, the next step is to integrate the model into our Open-Day light controller.

1. Environment Setup
   In the virtual implementation of the entropy-based detection system, the environment will first need to be configured. The steps are presented below;
   a. Virtual Box installation
   b. Ubuntu installation (1)
   c. Ubuntu Installation (2)
   d. RYU installation
   e. Mininet installation

f. Python 3.10 installation

The virtual box will be installed on a Windows 11 machine, it will be the box that houses the other installations.

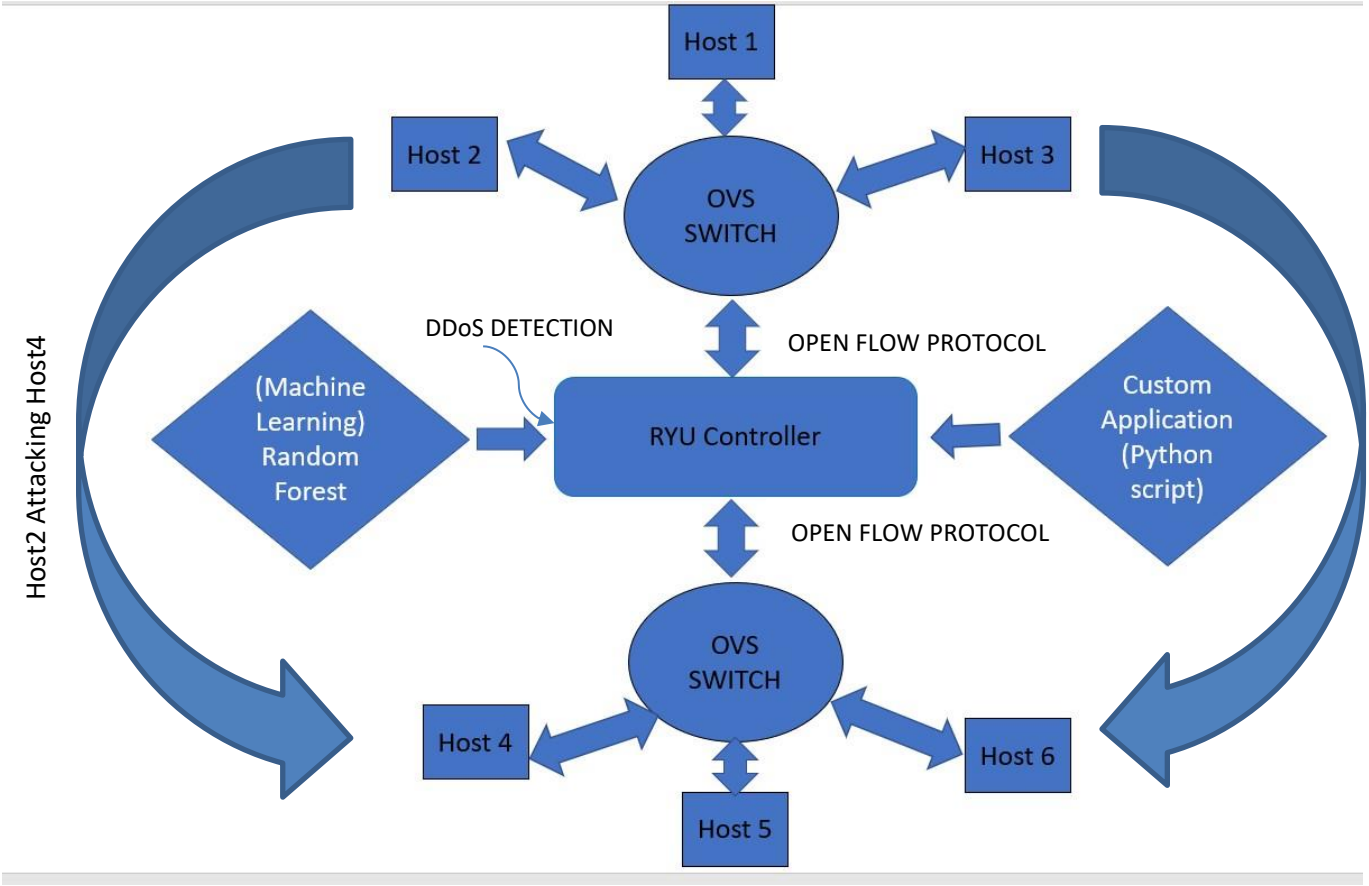
The RYU software will be installed in the first installation of Ubuntu while Mininet will be installed on the other. This is done so that we can truly visualize the implementation. Python 3.10 will be installed on both Ubuntu installations.

2. RYU Controller

The Random Forest model performed the best during the machine modelling phase so in building the custom application that would be integrated into the SDN environment, the RF model would be implemented in the application. The custom application will facilitate the interaction with Ryu controller enabling the incorporation of the machine learning model for the detection of DDoS traffic in the SDN network.

The custom application will be equipped with the ability to perform real-time analysis of network traffic. When data flows through the SDN network, the custom application that has be integrated into the Ryu controller enables it to monitor and evaluate the traffic for any indication of potential DDoS attack.

Fig. 5.11. Shows the diagrammatic description of 5.3.



CHAPTER 6: EVALAUATION OF WORK

6.1. INTRODUCTION

This project was able to successfully leverage entropy-based methods together with machine learning to detect DDoS traffic. The use of entropy-based method which involves the collection of traffic flow statistics from open- flow networks was used to generate the data which was then used in training different machine models and they all performed well but the Random-Forest machine performed the best with an accuracy of 100%. Entropy based systems are able to detect an anomaly in flow traffic i.e., it is able to detect when there is a significant shift in the normal flow of traffic in the SDN network.

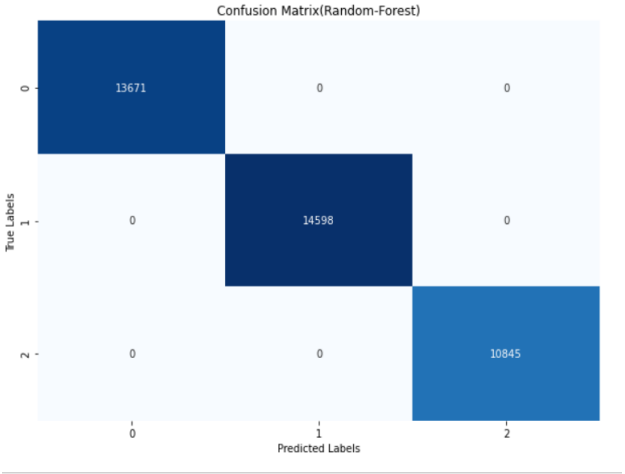
6.2. RESULTS

1. Machine Learning Model Training

   A total of three (3) models were trained with the well preprocessed dataset. The models are Random Forest (RF), k-Nearest Neighbor (KNN) and the Naïve Bayes. The accuracy, F1-score, precision and recall were used to evaluate the performance of each model. Below are diagrams showing the performance of each model;

   a. Random Forest

   Fig.6.2.1. shows the performance of the Random Forest model




```
In [38]: from sklearn.metrics import classification_report

         # Make predictions on the test set
         y_pred = rf_classifier.predict(X_test)

         # Calculate Precision, Recall, and F1-Score
         report = classification_report(y_test, y_pred)

         print("Classification Report:")
         print(report)
```

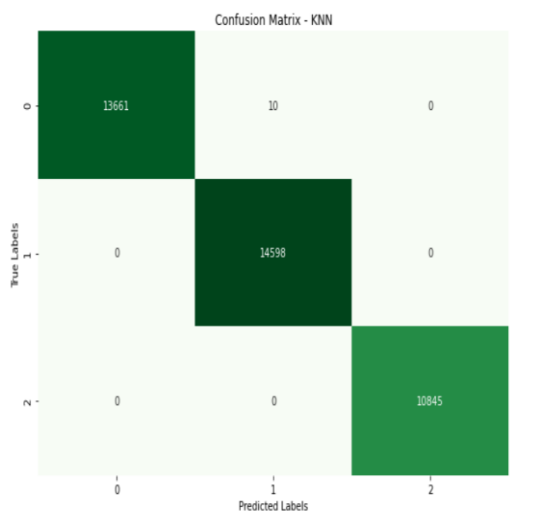
```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     13671
         1.0       1.00      1.00      1.00     14598
         2.0       1.00      1.00      1.00     10845

    accuracy                           1.00     39114
   macro avg       1.00      1.00      1.00     39114
weighted avg       1.00      1.00      1.00     39114
```

   b. K-Nearest Neighbor

   Fig.6.2.2. shows the performance of the KNN model




```
In [54]: from sklearn.metrics import classification_report

         # Make predictions on the test set
         y_pred = knn.predict(X_test)

         # Calculate Precision, Recall, and F1-Score
         report = classification_report(y_test, y_pred)

         print("Classification Report:")
         print(report)
```

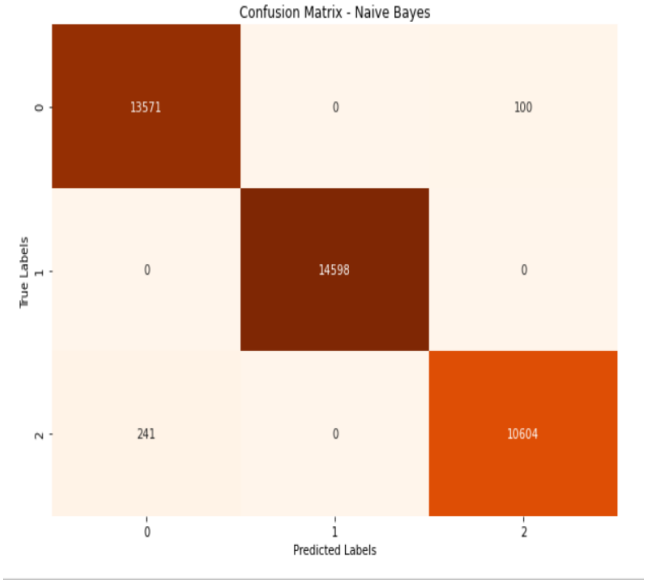
```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     13671
         1.0       1.00      1.00      1.00     14598
         2.0       1.00      1.00      1.00     10845

    accuracy                           1.00     39114
   macro avg       1.00      1.00      1.00     39114
weighted avg       1.00      1.00      1.00     39114
```

c. Naïve Bayes

Fig.6.2.3. shows the performance of the Naïve Bayes model




```
In [58]: # Make predictions on the test set
y_pred = naive_bayes_classifier.predict(X_test)

# Calculate Precision, Recall, and F1-Score
report = classification_report(y_test, y_pred)

print("Classification Report:")
print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.99     13671
         1.0       1.00      1.00      1.00     14598
         2.0       0.99      0.98      0.98     10845

    accuracy                           0.99     39114
   macro avg       0.99      0.99      0.99     39114
weighted avg       0.99      0.99      0.99     39114
```

The evaluations that was performed on all the trained machine learning models, was thorough as demonstrated by all the diagrams above. It can therefore be concluded that the Random-Forest did perform the best. This is the model that will be implemented in our SDN controller.

2. SDN Network Implementation

The SDN network was implemented using Mininet vm, Ryu controller vm, Hping3 tool, and Python 3. Mininet was was used for the creation of SDN controllers and switches which was used in testing the different network flows, routing algorithms and traffic management strategies. Ryu is a popular open-source SDN controller that allows you to control the behavior of SDN switches in a network. Hping3 was used to generate the desired DDoS attacks. Below are the follo