

DANMARKS TEKNISKE UNIVERSITET



## MASTER'S THESIS

---

**Advancing Security in the Era of Dynamic Networks: A Comprehensive Approach to Network Security with Software-Defined Networking and Programmable Data Planes**

---

**Usama Yasin**

**Supervisor:** Henrik Wessing

**Co-Supervisors:** Eder Ollora Zaballa (Postdoctoral Researcher, Ph.D)  
Mingyuan Zang (Postdoctoral Researcher, Ph.D)

## Abstract

The increasing sophistication of Distributed Denial of Service (DDoS) attacks challenges traditional network infrastructures, which often lack the flexibility and scalability to respond effectively. This thesis examines the integration of Software-Defined Networking (SDN), programmable data planes, and machine learning (ML) to enhance real-time DDoS detection and mitigation in wide-area networks. Two solutions are explored: an ONOS-based SDN approach using Machine Learning and a Java application for network control which serves as base line solution, and a P4-based solution with a programmable switch and Python control plane integrated with ML models. The ONOS solution offers dynamic rule updates and flexible integration of detection algorithms, while the P4-based solution provides low-latency, real-time threat detection directly at the data plane. Both solutions are tested in simulated environments using Mininet and evaluated on detection accuracy, efficiency, scalability, and flexibility. Results show that the ONOS-based solution is more adaptable for managing detection scripts and flow rules, whereas the P4-based solution excels in immediate threat detection and mitigation due to its embedded ML capabilities. This research demonstrates the potential of combining SDN, programmable data planes, and ML to create robust, scalable, and responsive network security solutions. Future work will focus on optimizing resource management, improving model training, and conducting real-world testing to further enhance these approaches.

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Background and Motivation . . . . .	7
1.2	Problem Statement . . . . .	7
1.3	Objectives . . . . .	8
1.4	Structure . . . . .	8
<b>2</b>	<b>Literature Review</b>	<b>9</b>
2.1	Network Security Challenges . . . . .	9
2.1.1	Software-Defined Networking (SDN) Security Concerns . . . . .	9
2.2	Overview of DDoS Attacks . . . . .	11
2.3	Traditional Network . . . . .	12
2.4	Software Defined Networking (SDN) and Programmable Data Planes (p4) . .	13
2.5	Machine learning in Programmable Data Planes (p4) . . . . .	15
2.6	Existing Solutions . . . . .	16
<b>3</b>	<b>Methodology</b>	<b>17</b>
3.1	Data Preparation and pre-processing . . . . .	17
3.2	Machine Learning Model Development . . . . .	17
3.3	DDoS Detection Solution Implementation . . . . .	17
3.4	System Testing and Evaluation . . . . .	18
<b>4</b>	<b>Dataset and Data Conversion Tool</b>	<b>18</b>
4.1	Feature Extraction and Conversion using CICFlowMeter . . . . .	19
<b>5</b>	<b>Machine Learning Model Selection</b>	<b>20</b>
5.1	Data pre-processing . . . . .	20
5.2	Feature Selection . . . . .	21
5.3	Model Training and Validation . . . . .	21
5.4	Performance Metrics . . . . .	22
5.5	Model Evaluation . . . . .	22
5.6	Feature Importance . . . . .	23
<b>6</b>	<b>ONOS-Based Solution</b>	<b>24</b>
6.1	Architecture Overview . . . . .	24
6.2	Workflow and Implementation . . . . .	24
6.3	Data Flow Diagram . . . . .	26
6.4	Constants and Configuration . . . . .	26
6.4.1	Key Constants . . . . .	26
6.4.2	Configuration Details . . . . .	28

6.5	Core Functionality: Methods in the Application . . . . .	29
6.5.1	main(String[] args) . . . . .	29
6.5.2	runPythonScript(String command, int timeoutSeconds) . . . . .	29
6.5.3	createBlocklistCSV(String scriptOutput, String csvFilePath) . . . . .	30
6.5.4	applyBlocklist(String csvFilePath) . . . . .	30
6.5.5	addBlockRuleToOnos(String srcIp) . . . . .	31
6.5.6	buildJsonPayload(String srcIp) . . . . .	31
6.5.7	fetchFromOnos(String endpoint) . . . . .	31
6.5.8	encodeCredentials(String username, String password) . . . . .	32
6.6	DDoS Attack Simulation . . . . .	32
6.7	Deployment and Testing Environment . . . . .	33
6.7.1	Hardware and Virtualization Environment . . . . .	33
6.7.2	Software Components . . . . .	33
<b>7</b>	<b>P4-Based Solution</b>	<b>34</b>
7.1	Development of P4 Application . . . . .	34
7.1.1	Header Definitions . . . . .	34
7.1.2	Metadata and Header Structures . . . . .	35
7.1.3	Packet Parser . . . . .	35
7.1.4	Ingress Processing . . . . .	35
7.1.5	Egress and Checksum Controls . . . . .	36
7.1.6	Deparser . . . . .	36
7.1.7	Main Control Block . . . . .	36
7.2	Python Control Plane Application . . . . .	37
7.2.1	Setup and Configuration . . . . .	37
7.2.2	Machine Learning Model Integration . . . . .	37
7.2.3	Data Loading and Preparation . . . . .	38
7.2.4	Switch Connection and Pipeline Configuration . . . . .	38
7.2.5	Table Management and ARP Handling . . . . .	38
7.2.6	DDoS Detection and ARP Entry Management . . . . .	39
7.2.7	Main Function and Application Execution . . . . .	39
7.3	Shutdown and Cleanup . . . . .	40
7.4	System Architecture and Data Flow . . . . .	40
7.4.1	Introduction to System Architecture . . . . .	40
7.4.2	Overview of Components . . . . .	40
7.4.3	System Workflow and Data Flow . . . . .	41
7.4.4	Data Flow Diagram . . . . .	42
7.4.5	Scalability and Flexibility of the Architecture . . . . .	42
7.4.6	Challenges and Considerations . . . . .	43

7.5	Deployment and Testing Environment . . . . .	43
7.5.1	Hardware and Virtualization Environment . . . . .	43
7.5.2	Software Components . . . . .	44
7.5.3	Machine Learning and Control Plane Libraries . . . . .	44
7.5.4	Network Topology Configuration . . . . .	45
7.5.5	Testing Procedures . . . . .	45
7.5.6	Environment Logs and Monitoring . . . . .	46
<b>8</b>	<b>Results</b>	<b>46</b>
8.1	DDOS Detection Results from ONOS solution . . . . .	46
8.2	DDOS Detection Results from p4 solution . . . . .	48
<b>9</b>	<b>Discussion</b>	<b>51</b>
9.1	Achieved Results Interpretation . . . . .	51
9.1.1	Detection Accuracy and Granularity . . . . .	51
9.1.2	Operational Efficiency . . . . .	51
9.1.3	Flexibility and Control . . . . .	51
9.1.4	Scalability and Maintenance . . . . .	52
9.1.5	Overall Effectiveness . . . . .	52
9.2	Implementation of the P4 Solution in a Real-World Scenario . . . . .	52
9.2.1	Deployment in Wide-Area Networks (WANs) . . . . .	52
9.2.2	Integration with Existing Network Infrastructure . . . . .	53
9.2.3	Dynamic Traffic Management and DDoS Mitigation . . . . .	53
9.2.4	Advantages . . . . .	53
9.2.5	Drawbacks and Challenges . . . . .	54
9.3	Does the Solution Answer the Research Question? . . . . .	54
9.3.1	Implications for Network Security . . . . .	55
9.4	Comparison with Existing Solutions . . . . .	56
9.5	Potential Improvements and Future Work . . . . .	56
<b>10</b>	<b>Conclusion</b>	<b>57</b>
	<b>References</b>	<b>60</b>

## Acronyms

<b>SDN</b>	Software-Defined Networking
<b>DDoS</b>	Distributed Denial of Service
<b>ML</b>	Machine Learning
<b>P4</b>	Programming Protocol-Independent Packet Processors
<b>IoT</b>	Internet of Things
<b>ONOS</b>	Open Network Operating System
<b>CSV</b>	Comma-Separated Values
<b>ARP</b>	Address Resolution Protocol
<b>API</b>	Application Programming Interface
<b>PCAP</b>	Packet Capture
<b>AUC-ROC</b>	Area Under the Receiver Operating Characteristic Curve
<b>TCP</b>	Transmission Control Protocol
<b>RF</b>	Random Forest
<b>DAD</b>	DDoS Attack Detection
<b>SYN</b>	Synchronize
<b>HTTP</b>	HyperText Transfer Protocol
<b>JSON</b>	JavaScript Object Notation
<b>OEMs</b>	Original Equipment Manufacturers
<b>URL</b>	Uniform Resource Locator
<b>CPU</b>	Central Processing Unit
<b>RAM</b>	Random Access Memory
<b>VM</b>	Virtual Machine
<b>MAC</b>	Media Access Control
<b>UDP</b>	User Datagram Protocol

**NaN**    Not a Number

**BMv2**   Behavioral Model version 2

**CLI**    Command Line Interface

# 1 Introduction

## 1.1 Background and Motivation

In recent years, the landscape of network security has been rapidly evolving due to the increasing complexity and advancement in network architectures. Traditional network infrastructures, which often rely on very strict non flexible and hardware-based configurations, are struggling to keep up with the demands for flexibility, scalability, and rapid deployment of new services. This evolution has introduced novel security challenges, particularly with the proliferation of sophisticated cyber-attacks such as Distributed Denial of Service (DDoS) attacks. These attacks can cripple networks by overwhelming them with malicious traffic, leading to significant financial and operational damages.

To address the challenges of traditional networks, new paradigms in networking, such as Software-Defined Networking (SDN) and programmable data planes (e.g., P4), have emerged. SDN decouples the control plane from the data plane, allowing for centralized and programmable control over the network. This enables more agile and adaptive network management and security policies. On the other hand, programmable data planes like P4 allow for fine-grained control over how packets are processed within the network, facilitating the implementation of custom and dynamic security measures directly in the network infrastructure.

Machine learning (ML) has also been recognized as a powerful tool in network security, offering the ability to detect and mitigate attacks in real-time by learning from enormous amounts of network traffic data. The convergence of SDN, programmable data planes, and ML presents a promising approach to enhancing network security in dynamic network environments.

## 1.2 Problem Statement

How can the integration of Software-Defined Networking (SDN), programmable data planes, and machine learning be effectively utilized to enhance real-time detection and mitigation of sophisticated DDoS attacks in real world wide-area networks, while overcoming the challenges of seamless integration, timely response, and system scalability?

This research question encapsulates the core focus of the thesis, highlighting the goal of leveraging modern technologies to improve network security and the specific challenges that need to be addressed.



### 1.3 Objectives

The primary objectives of this thesis are:

1. To develop a machine learning-based model for detecting DDoS attacks using the CIC-DDoS2019 dataset.
2. To integrate the trained ML model into a Java application that interacts with an ONOS SDN controller for DDoS detection to setup a base solution.
3. To simulate a DDoS attack in a Mininet network topology and collect network traffic data for analysis.
4. To develop a P4 application that utilizes a Python-based control plane to forward network traffic and detect DDoS attacks using the ML model.
5. To evaluate the performance and effectiveness of the proposed approach in detecting and mitigating DDoS attacks in a dynamic network environment.

### 1.4 Structure

This thesis is structured into following chapters:

1. **Introduction:** This chapter provides an overview of the background and motivation for the research, outlines the problem statement, sets the objectives of the study, and presents the structure of the thesis.
2. **Literature Review:** This chapter reviews the existing literature on network security challenges, including specific issues related to Software-Defined Networking (SDN). It also provides an overview of DDoS attacks and discusses traditional network security solutions, the role of SDN and programmable data planes, and the application of machine learning in programmable data planes.
3. **Methodology:** This chapter outlines the research methodology, including data preparation and pre-processing, machine learning model development, DDoS detection solution implementation, and system testing and evaluation.
4. **Dataset and Data Conversion Tool:** This chapter details the data collection and conversion processes, focusing on the use of the CICFlowMeter tool for extracting features from network traffic and creating datasets for machine learning.
5. **Machine Learning Model Selection:** This chapter discusses the selection and development of the Random Forest model for DDoS detection, including data pre-processing, feature selection, model training and validation, performance metrics, model evaluation, and the importance of features.

6. **ONOS-Based Solution:** This chapter describes the architecture, workflow, and implementation of the ONOS-based DDoS detection solution. It covers the key constants and configuration settings, the core functionality of the methods used in the Java application, the DDoS attack simulation process, and the evaluation of the solution's effectiveness.
7. **P4-Based Solution:** This chapter explains the development and implementation of the P4-based solution for DDoS detection, including the Python control plane application, system architecture, data flow, deployment, and testing environment. It also discusses the scalability, flexibility, challenges, and considerations of the solution.
8. **Results:** This chapter presents the results of the DDoS detection experiments conducted using both the ONOS-based and P4-based solutions. It compares the performance, accuracy, and efficiency of the two approaches in detecting and mitigating DDoS attacks.
9. **Discussion:** This chapter interprets the results obtained from the experiments. It examines the implications for network security, discusses potential improvements, and suggests directions for future work.
10. **Conclusion:** This chapter summarizes the key findings of the thesis..

## 2 Literature Review

### 2.1 Network Security Challenges

Network security is a critical concern in today's interconnected world, where cyber threats are becoming increasingly sophisticated and prevalent. As networks evolve and new technologies emerge, so do the challenges in safeguarding them. This section aims to provide an overview of network security challenges, drawing upon insights from recent research and advancements in the field.

#### 2.1.1 Software-Defined Networking (SDN) Security Concerns

Software-Defined Networking (SDN) brings significant advantages in terms of network flexibility, programmability, and centralized management, but it also introduces specific security vulnerabilities that are not present in traditional network architectures. The centralized controller in an SDN environment functions as the network's "brain," managing all network operations, including forwarding decisions and traffic management. This centralization, while beneficial for network efficiency and ease of management, creates a single point of failure and a highly attractive target for attackers. If an attacker successfully compromises the SDN controller, they can potentially gain control over the entire network. This could lead to

catastrophic consequences, such as unauthorized access to network resources, data breaches, and manipulation of network traffic flows. Distributed Denial of Service (DDoS) attacks represent a particularly severe threat in this context. By overwhelming the SDN controller with a flood of traffic, these attacks can exhaust the controller's resources, rendering it unable to process legitimate network requests and leading to widespread network outages and service disruptions. Moreover, the control plane's centralized nature is susceptible to a variety of sophisticated attacks. Controller spoofing, for instance, involves an attacker masquerading as the legitimate controller to intercept or manipulate communication between the controller and the data plane devices. Flow rule manipulation and injection attacks are other serious concerns; an attacker could potentially alter flow rules or insert malicious ones, redirecting or blocking traffic in ways that could lead to data loss, espionage, or further spreading of malware. These vulnerabilities underscore the critical need for robust security measures and vigilant monitoring within SDN environments to safeguard against these unique risks [11].

"Security in Software-Defined-Networking: A Survey," by Yao and Yan (2016) [20], provides an extensive review of the security concerns within Software-Defined Networking (SDN). While SDN offers flexibility and centralized management, it also introduces several security vulnerabilities. The paper identifies three primary security challenges in SDN: network intrusion, denial of service (DoS) and distributed denial of service (DDoS) attacks, and application trust management. Network intrusion remains a critical issue due to SDN's architecture, where attackers can gain unauthorized access to the control plane or the application plane, leading to data breaches or malicious activities. The centralized nature of SDN control makes it highly susceptible to DoS/DDoS attacks, where attackers can overwhelm controllers with excessive flow requests, potentially leading to network outages. Furthermore, the programmable nature of SDN allows for the rapid deployment of new applications, but this also raises concerns about the trustworthiness and security of these applications, as malicious or poorly designed applications could compromise the entire network. The paper emphasizes the need for fine-grained access control, robust authentication mechanisms, and self-healing capabilities to address these challenges. It suggests that future research should focus on developing comprehensive security frameworks that integrate these features to mitigate risks effectively in SDN environments. This survey is particularly relevant to my research as it highlights the inherent security risks in SDN that must be considered when integrating SDN with other technologies like programmable data planes and machine learning for real-time network defense.

## 2.2 Overview of DDoS Attacks

"In the evolving landscape of cyber threats, Distributed Denial of Service (DDoS) attacks have become increasingly sophisticated, exploiting vulnerabilities in network design and security. Shui Yu, in "Distributed Denial of Service Attack and Defense," outlines the historical progression and escalating complexity of DDoS attacks, emphasizing that the inherent insecurity in the internet's original architecture has left cyberspace vulnerable to such threats. Yu discusses the formation of botnets and advanced evasion techniques like fast flux and domain flux, which complicate detection efforts. This historical perspective is crucial for understanding the development of more effective defense mechanisms against DDoS attacks [21].

Building on this, Guo, Jing, and Zhao (2023) explore DDoS attack detection and defense within Software-Defined Networking (SDN) environments. They categorize detection methods into mathematical statistics, machine learning, and novel architectures, underscoring the inherent vulnerabilities of SDN's centralized control plane to DDoS attacks. The paper advocates for a hybrid approach that combines coarse-grained traffic classification with machine learning for detailed analysis, aligning with the trend towards more adaptive and intelligent security measures. This research is particularly relevant to my work, which leverages SDN and programmable networks to enhance DDoS detection mechanisms [6].

Further exploring SDN vulnerabilities, Bikbulatov and Kurochkin (2019) simulate DDoS attacks on SDN controllers using Mininet, demonstrating how easily these controllers can be overwhelmed. Their findings highlight the necessity for robust security measures within SDN architectures to prevent network paralysis under attack. This study's insights into the simulation of DDoS scenarios provide a foundation for developing more resilient SDN and programmable network architectures, such as those utilizing P4 [7].

Complementing these studies, Sanmorino and Yazid (2013) propose the DASHM framework for real-time DDoS detection and mitigation using machine learning. Their approach, which employs a Self-Organizing Map (SOM) algorithm to classify network traffic and identify attacks, is significant for its dynamic response capability, adjusting to real-time traffic conditions. This framework's use of machine learning within programmable environments resonates with my research focus, offering a model for integrating intelligent threat detection directly into network infrastructure [19].

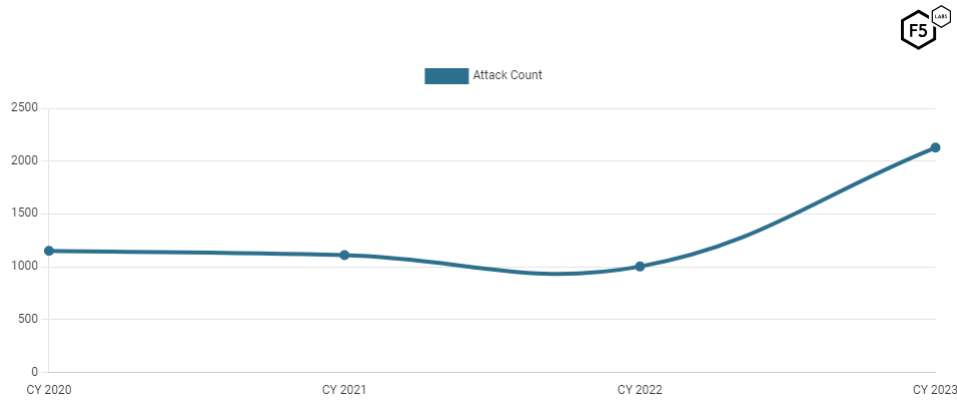


Figure 1: DDoS Trend

In figure 1, DDoS attacks more than doubled over 2023, exploding from just over 1,000 in 2022 to more than 2,100 a year later [2].

## 2.3 Traditional Network

Bholebawa, Jha, and Dalal (2016) conduct a comparative performance analysis between traditional IP-based networks and OpenFlow-enabled networks, highlighting the complexities and limitations of traditional network architectures. Traditional networks are characterized by vertical integration, where control and data planes are bundled within core network devices, such as routers and switches. This tight integration results in networks that are complex, difficult to manage, and less flexible, often requiring manual configuration of individual devices. Such manual configurations can be time-consuming and prone to errors, hindering the scalability and adaptability necessary to meet the growing demands of modern digital applications and users. Additionally, traditional networks follow specific rules set by Original Equipment Manufacturers (OEMs), are often statically configured, and lack support for prioritization of packets. This inflexibility leads to a more structurally complex setup, with high costs associated with management and reconfiguration. Once a configuration is implemented, it becomes quite challenging to modify it, thereby limiting the network's responsiveness to change [12].

In "Comparative Analysis of Traditional and Software Defined Networks," Muhammad Awais et al. (2021) present a thorough comparative study of traditional IP networks and Software-Defined Networks (SDN). The study highlights several key differences between these two networking paradigms, focusing particularly on architecture, protocols, and management capabilities. In traditional networks, the integration of the control and data planes within network devices leads to static and inflexible configurations. These configurations rely heav-

ily on manual management and are prone to errors. The complexity of managing a large-scale traditional network increases with the diversity of devices and protocols, making it difficult to implement dynamic policies or respond rapidly to network changes. This setup results in significant configuration overhead and limited support for traffic engineering and dynamic scalability [15].

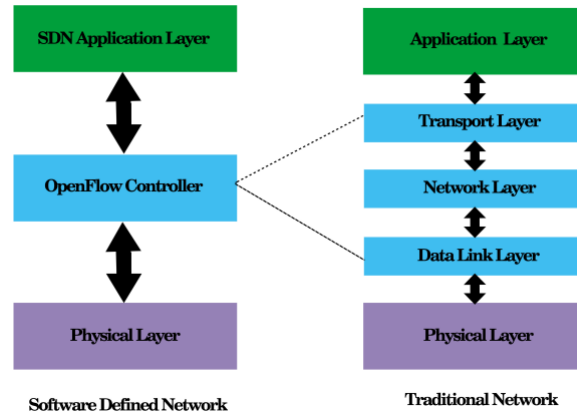


Figure 2: SDN vs Traditional Architecture [15]

In contrast, SDN separates the control plane from the data plane, centralizing network intelligence in a programmable controller that dynamically manages network devices through standardized protocols like OpenFlow. This decoupling allows for more flexible and scalable network management, enabling automated configuration, efficient fault management, and easier implementation of new network policies. SDN's programmable nature enhances network agility and supports advanced traffic engineering capabilities, positioning it as a more adaptable and efficient solution for modern networking challenges.

## 2.4 Software Defined Networking (SDN) and Programmable Data Planes (p4)

Software Defined Networking (SDN) is a network architecture that separates the control plane from the data plane. This decoupling allows for centralized control and management of the network, enabling more flexible, efficient, and programmable network operations. SDN's centralized controller manages and configures network devices, improving resource utilization, simplifying network management, reducing operational costs, and fostering innovation and evolution in network design. Unlike traditional IP-based networks, which are complex and difficult to manage due to the vertical integration of control and data planes within network devices, SDN offers a more agile and adaptable approach. The implementation of SDN, particularly through protocols like OpenFlow, allows for dynamic and programmable control over network traffic, paving the way for advanced features like multi-path routing and

enhanced network performance. This shift towards SDN represents a significant advancement in addressing the limitations of traditional network architectures, positioning SDN as a crucial development in the future of networking [22].

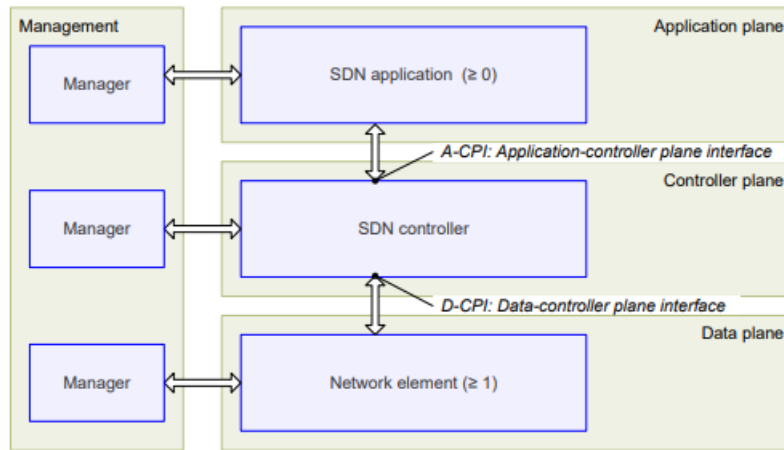


Figure 3: SDN overview with physical data plane  
[17]

In "A Taxonomy-Based Approach for Security in Software-Defined Networking," Banse and Schuette (2017) [9] present a novel framework for enhancing security in Software-Defined Networking (SDN) environments. The authors introduce a taxonomy-based policy engine that uses first-order logic to define fine-grained security policies. This approach addresses the challenges of dynamically controlling access to critical network resources, particularly in multi-tenant environments where entities are not known at the time of network planning. The taxonomy provides a structured classification of SDN network entities, allowing for more flexible and dynamic policy definitions compared to traditional static rule sets. By integrating a Prolog-based policy engine, the framework allows for real-time policy decision-making, which can be crucial for maintaining security in rapidly evolving network environments. This study is relevant to my research on P4 and SDN security as it provides insights into implementing flexible security mechanisms that can adapt to changing network conditions, enhancing the understanding of how to manage access control in programmable network infrastructures effectively.

"Programming Protocol-Independent Packet Processors," Bosshart et al. (2014) introduce P4, a high-level programming language designed to configure protocol-independent packet processors. Traditional SDN approaches, such as those based on OpenFlow, have become increasingly complex due to the need to support a growing number of protocol headers. P4 addresses this complexity by allowing network operators to define how packets should be processed without being tied to specific protocols or hardware implementations. The key

goals of P4 include reconfigurability, protocol independence, and target independence. This flexibility allows P4 to support a wide range of network protocols and to adapt to new protocols without requiring hardware changes. P4 raises the level of abstraction for programming network devices, making it easier to implement and manage sophisticated network functions dynamically, which is essential for the evolution of SDN and the development of more adaptable and efficient network infrastructures [8].

"A Framework for In-network Inference using P4," Wang et al. (2021) [3] proposes an innovative approach to integrating machine learning inference capabilities directly within network devices using P4, a programmable data plane language. Traditional network security models often rely on centralized processing, which can introduce latency and bottlenecks. This framework leverages P4's flexibility to implement decision tree models within the switch itself, enabling real-time anomaly detection and traffic classification without needing data to be transferred to external servers. The research highlights the efficiency and speed of in-network processing for security applications, demonstrating that embedding ML models within programmable switches can significantly reduce response times and improve network resilience against attacks. This work provides a valuable foundation for developing real-time DDoS detection mechanisms, as explored in my research, by illustrating how P4 can be utilized to enhance the capabilities of network security systems through programmable data planes.

## 2.5 Machine learning in Programmable Data Planes (p4)

Coelho and Schaeffer-Filho (2022) propose a novel system called BACKORDERS, which leverages Random Forests (RF) for detecting Distributed Denial of Service (DDoS) attacks in programmable data planes. Their approach involves embedding RF models within programmable switches to classify network traffic at line rate, thereby ensuring high accuracy and efficiency. The system uses a feature extraction module to compute relevant statistics from network packets and an online classifier module to orchestrate multiple classification trees for real-time traffic analysis. Evaluations using the CICIDS2017 dataset demonstrated that BACKORDERS could achieve F1-scores exceeding 90% with relatively low resource usage, highlighting its potential for effective DDoS detection in modern network infrastructures. This research underscores the significant advancements in using machine learning within programmable networks to enhance security measures [4].

Musumeci et al. (2022) present a study on detecting Distributed Denial of Service (DDoS) attacks in Software Defined Networking (SDN) environments using Machine Learning (ML) algorithms within P4 programmable networks. The research focuses on Transmission Control Protocol (TCP) SYN flood attacks and explores two DDoS Attack Detection (DAD) architectures: Standalone DAD, where detection occurs locally at each network switch, and



Correlated DAD, where detection is centralized. By leveraging the P4 language, the study demonstrates the ability to perform real-time DAD with high accuracy, precision, recall, and F1-scores, all exceeding 98%. The results show significant latency reduction when features are extracted directly at the data plane, highlighting the efficiency and effectiveness of combining ML with programmable data planes for DDoS detection [16].

## 2.6 Existing Solutions

There are 2 solutions that i have found are relevant in terms of what i am trying to accomplish with my research. The 3D-PM solution is one such approach that aims to enhance the detection of DDoS attacks by utilizing P4 programmable switches combined with machine learning models. The system operates at the data plane level, allowing for direct detection of attacks without overburdening the SDN controller. By using ML models like Random Forest, trained on realistic traffic datasets, 3D-PM achieves high detection accuracy (up to 98.95%) and effectively identifies various types of flooding attacks, such as ICMP, UDP, TCP SYN, and HTTP GET floods. The solution also employs a probabilistic scoring mechanism to improve its robustness against variable-rate attacks, enabling more accurate and timely alerts. While 3D-PM demonstrates the potential of integrating P4 programmability with ML for real-time attack detection, it has limitations in terms of scalability and adaptability. It primarily focuses on specific types of flood attacks and uses a probabilistic scoring method to manage dynamic attack rates. While this approach effectively reduces controller overload and provides real-time detection, it does not fully address scalability across larger, more diverse network environments or a wider variety of DDoS attack types.[10].

Another notable approach is DataPlane-ML, which utilizes white-box switches enhanced with P4 constructs to run machine learning algorithms directly on the data plane. This solution aims to reduce the load on the SDN controller by enabling real-time DDoS detection closer to the source of the attack. By employing algorithms like K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Random Forest (RF), DataPlane-ML improves detection speed and accuracy compared to traditional statistical methods. It provides a mechanism that not only detects SYN Flood attacks but can also be extended to handle other types of DDoS attacks. DataPlane-ML showcases the feasibility of executing ML algorithms at the data plane, presenting an alternative to the centralized approaches commonly found in SDN environments. However, this approach also faces challenges, particularly in managing the computational complexity and resource constraints of implementing ML at the data plane level. Additionally, while DataPlane-ML reduces the burden on the controller, the need for significant computational resources within the data plane devices themselves could limit its applicability in highly resource-constrained environments [18].

Both 3D-PM and DataPlane-ML show promise in improving detection accuracy and response times, but their scalability in larger, more complex network environments remains a concern. These solutions may face challenges in adapting to varied network topologies and traffic patterns, particularly as network size and diversity increase. The focus of existing solutions on specific attack types (e.g., flood attacks in the case of 3D-PM) suggests a need for more comprehensive detection mechanisms that can handle a wider range of DDoS attack vectors, including both volumetric and non-volumetric attacks, with high accuracy. Implementing ML algorithms directly at the data plane, as seen in DataPlane-ML, can lead to significant resource usage, potentially straining switch capabilities. There is a need for solutions that balance high detection accuracy with efficient use of network resources, ensuring that performance remains robust even under high load conditions. In response to these gaps, the proposed p4 solution in this research aims to integrate Software-Defined Networking (SDN), programmable data planes, and machine learning in a more cohesive and scalable manner to enhance the real-time detection and mitigation of sophisticated DDoS attacks. By focusing on seamless integration between the control and data planes, my solution will aim to provide rapid, real-time responses to DDoS attacks without compromising the operational efficiency of the SDN controller or the underlying network infrastructure.

## 3 Methodology

### 3.1 Data Preparation and pre-processing

The CICDDoS2019 dataset from the Canadian Institute for Cybersecurity was utilized to train the machine learning model. This dataset includes a variety of DDoS attack types along with benign traffic, providing a realistic basis for model training. The dataset was pre-processed to handle missing values, normalize numerical features, and encode categorical variables, ensuring it was ready for machine learning applications.

### 3.2 Machine Learning Model Development

A Random Forest classifier was chosen as the machine learning model due to its robustness and effectiveness in handling high-dimensional data. The model was trained on the pre-processed CICDDoS2019 dataset. Key performance metrics, including accuracy, precision, recall, F1-score, and AUC-ROC, were used to evaluate the model's effectiveness.

### 3.3 DDoS Detection Solution Implementation

Two complementary DDoS detection solutions were developed:

**A: ONOS-Based Solution**

1. This solution was developed to serve as the baseline before developing the p4 solution.
2. A Docker container running ONOS (Open Network Operating System) was set up to manage the network topology.
3. DDoS attacks were simulated using hping, and the resulting traffic was captured in PCAP files.
4. The captured PCAP files were converted to CSV format using CICFlowMeter-V3 to ensure compatibility with the CICDDoS2019 dataset.
5. A Python application was developed to use the trained Random Forest model to predict DDoS attacks on the new data collected from the simulated environment. The predictions identified the source IP and port of the attacks.
6. A Java application was created to connect to ONOS, run the Python detection application, and based on the detection add rule to block the malicious ip address.

#### **B: P4-Based Solution**

1. P4 application was developed to handle packet processing and forwarding rules.
2. Python application was also developed which acted as the controller, setting the rules dynamically based on DDoS detection.
3. The Python app communicated with the P4 switches using gRPC to manage the packet forwarding rules.
4. The python control plane application performed DDoS detection and removed the ARP entries for malicious IP address to essentially drop all traffic from that IP address.

### **3.4 System Testing and Evaluation**

The integrated solutions were tested in a simulated network environment to evaluate their effectiveness in detecting and mitigating DDoS attacks. Performance metrics such as detection accuracy, false positive rate, and detection latency were analyzed. The results demonstrated the feasibility and efficiency of the combined use of machine learning and programmable data plane techniques for real-time DDoS detection and mitigation.

## **4 Dataset and Data Conversion Tool**

In this project, we utilized the CICDDoS2019 dataset from the Canadian Institute for Cybersecurity. This dataset was chosen due to its comprehensive and realistic representation

of Distributed Denial of Service (DDoS) attacks, which include a variety of modern reflective and exploitation-based attack types such as PortMap, NetBIOS, LDAP, MSSQL, UDP, UDP-Lag, SYN, NTP, DNS, and SNMP. The dataset's inclusion of both raw network traffic (PCAP files) and over 80 extracted network flow features makes it particularly valuable for developing and evaluating advanced DDoS detection methodologies. By using the CICDDoS2019 dataset, our project benefits from a realistic and diverse set of data that enhances the robustness and reliability of our detection models, ensuring their applicability to real-world scenarios [13].

#### 4.1 Feature Extraction and Conversion using CICFlowMeter

CICFlowMeter is a robust network traffic flow generator and analyzer designed to extract detailed features from network traffic. It was created by Canadian Institute for Cybersecurity and was used to create the CIC-DDoS2019 dataset. It facilitates the generation and analysis of bidirectional flows, where the directions are determined by the first packet of the flow—identifying the forward (from source to destination) and backward (from destination to source) directions. This capability allows CICFlowMeter to compute over 80 statistical network traffic features, such as Duration, Number of Packets, Number of Bytes, and Packet Length, separately for both forward and backward directions. CICFlowMeter offers additional functionalities, including the ability to select specific features from a comprehensive list, introduce new features, and control the flow timeout duration. The application outputs data in CSV format, which includes six essential columns for each flow: FlowID, SourceIP, DestinationIP, SourcePort, DestinationPort, and Protocol. Alongside these identifiers, the CSV file contains more than 80 features derived from network traffic analysis [1].

Figures 4 and 5 represent the layout of the CICFlowMeter program.

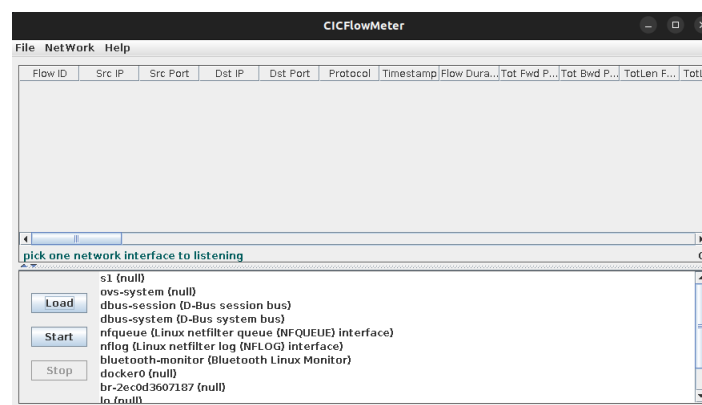


Figure 4: Real-time Conversion to csv

Figure 4 shows that it can be used to monitor and capture real time traffic on the available devices.

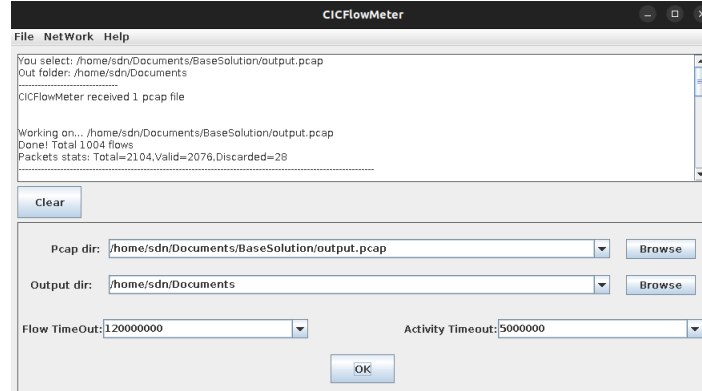


Figure 5: Conversion from pcap to csv

In figure 5 the tool can be used to extract features from a pcap file to csv file. This feature was used to extract data from the pcap file that was generated in the simulation of DDoS attack in our virtual environment.

It was essential to use the same tool, CICFlowMeter, for feature extraction from our DDoS simulations in both the ONOS DDoS solution and the P4 DDoS solution because by doing so it ensures consistency with the feature extraction process used to create the training dataset (CIC-DDoS2019). This consistency is critical for maintaining the accuracy of the machine learning model's predictions and avoiding potential issues related to incompatible features.

## 5 Machine Learning Model Selection

The selection of an appropriate machine learning model is crucial for achieving accurate and reliable predictions in any data-driven project. For this study, the Random Forest classifier was chosen due to its robustness, efficiency, and superior performance in handling large datasets with complex interactions among features. This section outlines the steps taken to preprocess the data, train the model, and evaluate its performance.

### 5.1 Data pre-processing

Initial pre-processing involved loading the data, handling missing values, and stripping leading and trailing spaces from column names to ensure consistency.

Key steps in pre-processing included:

1. Filling missing values in the 'Label' column with 'Normal'.

- 2. Removing unnecessary columns like 'Source IP' and 'Destination IP' to focus on relevant features.
- 3. Converting categorical data into numerical format using OneHotEncoder, which resulted in a sparse matrix for efficient storage and processing.

5.2 Feature Selection

Given the large number of features in the dataset, it was essential to select the most relevant ones to improve model performance and reduce computational complexity. Based on domain knowledge and preliminary analysis, the following top features were selected:

No.	Metric	Description
1	Flow Bytes/s	Total number of bytes transmitted per second during a flow
2	Flow Packets/s	Number of packets transmitted per second within a flow
3	Flow IAT Mean	Average time interval between packets in a flow
4	Flow IAT Std	Standard deviation of the inter-arrival times between packets in a flow
5	Flow IAT Max	Maximum time interval between two consecutive packets in a flow
6	Flow IAT Min	Minimum time interval between two consecutive packets in a flow

Table 1: Features and Descriptions

These features were chosen due to their significant impact on distinguishing between normal and attack traffic.

5.3 Model Training and Validation

The model was initialized with the following parameters:

No.	Parameter	Description
1	n_estimators=5	The number of trees in the forest
2	max_depth=10	The maximum depth of each tree to prevent over-fitting
3	random_state=0	For reproducibility
4	n_jobs=2	To leverage parallel processing for faster computation

Table 2: Random Forest Parameters and Descriptions

To evaluate the model’s performance, a 5-fold cross-validation was performed, yielding the following accuracy scores:

No.	Metric	Value
1	Cross-validation scores	[0.99932472, 0.99939435, 0.99934442, 0.99932866, 0.9993615]
2	Mean cross-validation score	0.9993507
3	Standard deviation	2.54e-05

Table 3: Cross-Validation Results

These results indicate that the model consistently achieves high accuracy across different folds, demonstrating its reliability and robustness.

## 5.4 Performance Metrics

The performance metrics used in this evaluation are precision, recall and F1-score:

$$\text{Precision} = \frac{TP}{(TP + FP)} \quad (1)$$

$$\text{Recall} = \frac{TP}{(TP + FN)} \quad (2)$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3)$$

## 5.5 Model Evaluation

Further evaluation metrics were computed on a separate test set to ensure the model's generalizability. The confusion matrix, precision, recall, and F1 score were calculated, revealing the following:

No.	Metric	Value
1	Precision	[0.9671017826156283]
2	Recall	[0.9148616270175567]
3	F1 Score	[0.9394325051670342]

Table 4: Performance Metrics

In figure 6, the ROC curve and the AUC demonstrate that the model used for detecting DDoS attacks is highly effective. It has perfect sensitivity (or recall) and specificity, meaning it correctly identifies all instances of DDoS attacks without falsely flagging any normal traffic.

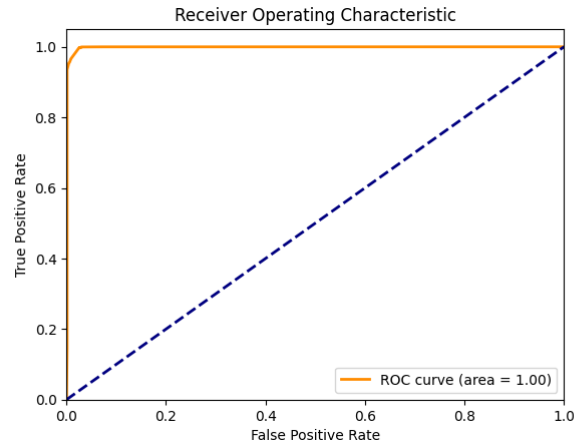


Figure 6: ROC Chart

## 5.6 Feature Importance

To understand the contribution of each feature, feature importance were extracted from the Random Forest model. The top features in terms of importance were:

No.	Metric	Value
1	Flow Bytes/s	0.38
2	Flow IAT Mean	0.26
3	Flow IAT Std	0.13
4	Flow IAT Max	0.11
5	Flow Packets/s	0.10
6	Flow IAT Min	0.02

Table 5: Feature Importance

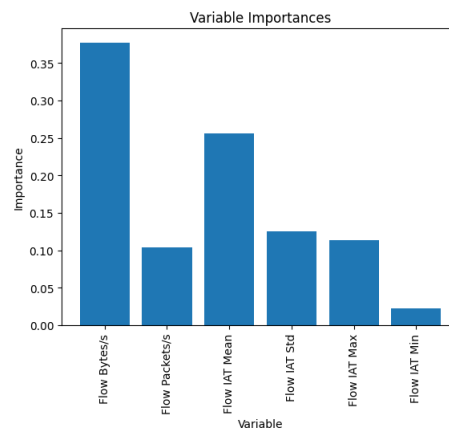


Figure 7: Feature Importance Graph

Figure 7 demonstrates a visual representation of feature importance.



## 6 ONOS-Based Solution

This section describes the development and functionality of a Java application designed to integrate DDoS detection with the Open Network Operating System (ONOS) controller. The application automates the process of detecting DDoS attacks, managing network security, and dynamically interacting with the ONOS controller through HTTP requests. This integration enhances network security by efficiently detecting and mitigating DDoS threats in a Software-Defined Networking (SDN) environment.

### 6.1 Architecture Overview

The proposed solution is designed around the following key components:

1. **ONOS Controller:** This serves as the central network controller responsible for managing the flow rules across the network switches. It interacts with network devices using the OpenFlow protocol, enabling fine-grained control over packet forwarding behaviors. ONOS also provides REST APIs, which are utilized to programmatically add or modify flow rules for blocking malicious traffic.
2. **Python-Based DDoS Detection Module:** A Python script (`ddos_detect.py`) is developed to analyze network traffic data and identify potential DDoS attack patterns. This script processes traffic logs or real-time data to compute metrics like source IP frequency, which helps in identifying IPs involved in anomalous traffic behavior indicative of DDoS attacks.
3. **Flow Rule Management Module:** This Java-based module interacts with the ONOS controller to enforce network policies based on the findings of the DDoS detection module. It uses the ONOS REST API to dynamically install flow rules that block malicious IP addresses identified by the Python script.
4. **Automation and Orchestration:** A comprehensive orchestration mechanism is implemented to automate the execution of the DDoS detection script, the processing of its output, and the application of necessary flow rules in the ONOS controller. This automation ensures timely response to detected threats without requiring manual intervention.

### 6.2 Workflow and Implementation

The development process is structured around several stages to ensure a robust and effective DDoS detection solution:

1. **Data Collection and Analysis:** The first step involves capturing network traffic data that is indicative of DDoS activity. The Python-based DDoS detection script

analyzes this data to identify patterns typical of such attacks, such as a sudden spike in traffic from specific source IPs.

2. **Integration with ONOS Controller:** The Java application integrates with the ONOS controller through RESTful APIs. It is responsible for fetching the network state and pushing flow rules to block identified malicious traffic sources. The communication with ONOS is secured using basic authentication, and all interactions are performed over HTTP.
3. **Dynamic Flow Rule Installation:** Once the Python script detects potential DDoS attack patterns, it outputs a list of malicious IP addresses along with their traffic counts. This output is processed by the Java application, which then dynamically constructs JSON payloads for ONOS REST API calls. The JSON payloads specify flow rules that drop traffic from these malicious IPs. This mechanism ensures that the network can adapt quickly to evolving threats.
4. **Automation of Detection and Mitigation:** The solution is designed to run periodically, leveraging a scheduled execution of the Python script. The results are automatically parsed, and the highest priority malicious IPs are selected for blocking. The Java application ensures that the necessary flow rules are added to the ONOS controller without human intervention, providing an automated defense mechanism against DDoS attacks.
5. **Error Handling and Logging:** Robust error handling is integrated throughout the Java application. If the Python script fails to execute or returns an error, the Java application captures this and logs it for future analysis. Similarly, any issues encountered while communicating with the ONOS controller, such as HTTP errors or connection timeouts, are also logged. This ensures that administrators are aware of any issues in real time and can take corrective actions.

## 6.3 Data Flow Diagram

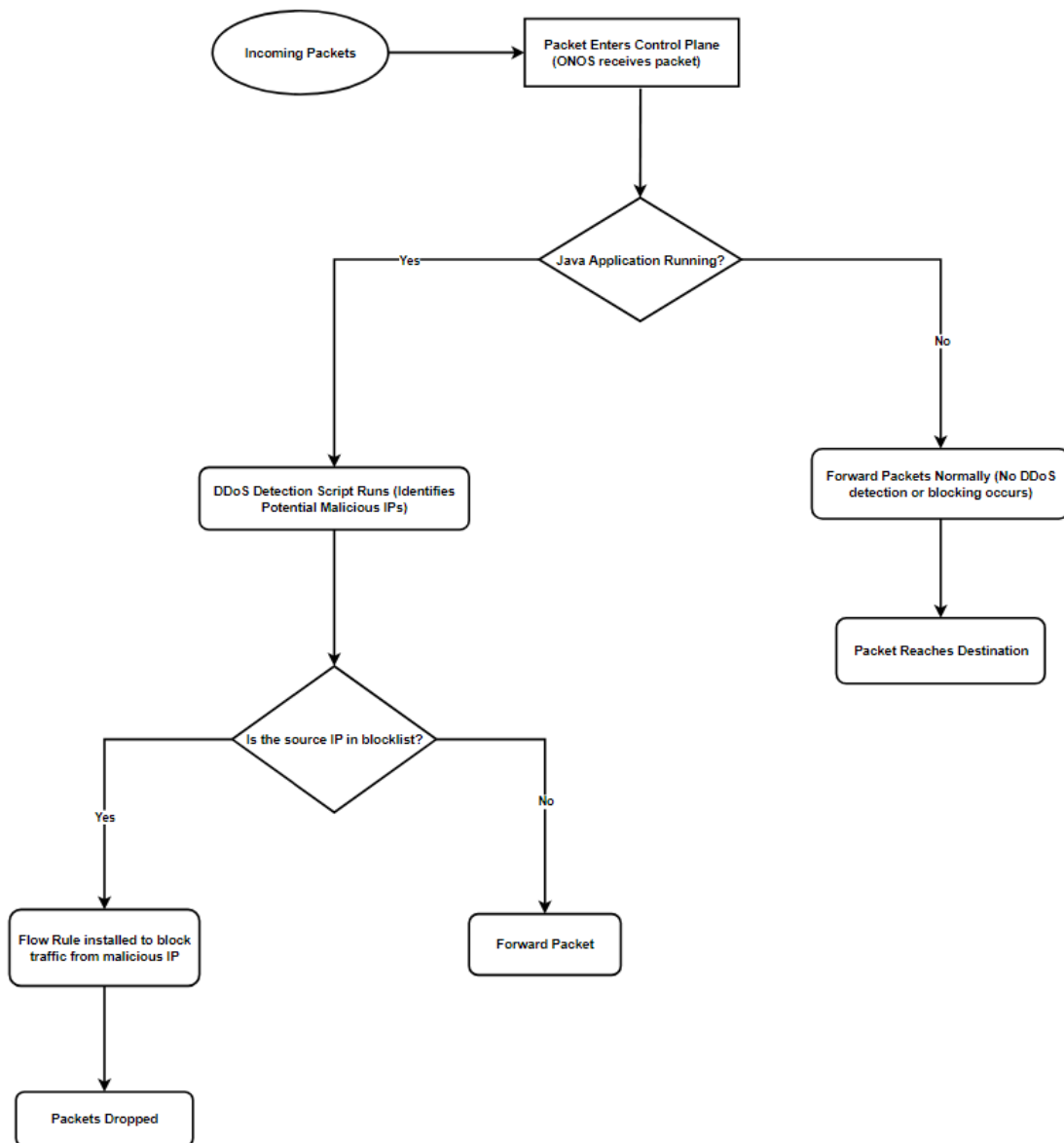


Figure 8: Onos Solution Data Flow

## 6.4 Constants and Configuration

This section discusses the constants used in the Java application, configuration settings, and how they influence the overall behavior of the DDoS detection and mitigation process.

### 6.4.1 Key Constants

#### 1 - ONOS Controller URL (ONOS\_URL):

- Definition: The base URL for the ONOS controller's REST A

- Example: "http://172.17.0.2:8181/onos/v1/"
- Purpose: This URL is used to send REST API requests to the ONOS controller for tasks such as retrieving network information and installing flow rules. It is crucial to set this URL correctly to ensure that the Java application can communicate with the ONOS controller.

## **2 - Authentication Credentials (USERNAME and PASSWORD):**

- Definition: The username and password required to authenticate with the ONOS REST API.
- Example: Username = "onos", Password = "rocks"
- Purpose: ONOS requires basic authentication for REST API access. These credentials are used to generate a Base64-encoded string for the Authorization header in HTTP requests, ensuring secure communication between the Java application and ONOS.

## **3 - Python Interpreter Path (PYTHON\_INTERPRETER):**

- Definition: The file path to the Python interpreter used to execute the DDoS detection script.
- Example: "/usr/bin/python3"
- Purpose: This path specifies which Python interpreter is used to run the DDoS detection script (ddos\_detect.py). Configuring the correct path is essential for the Java application to successfully invoke the script.

## **4 - DDoS Detection Script Path (DDOS\_DETECT\_SCRIPT):**

- Definition: The file path to the Python script that performs DDoS detection.
- Example: "/home/sdn/ddos\_detect.py"
- Purpose: This script analyzes network traffic data to identify potential DDoS attacks. The Java application runs this script periodically to detect and respond to DDoS threats.

## **5 - Script Execution Timeout (SCRIPT\_TIMEOUT):**

- Definition: The maximum amount of time (in seconds) allowed for the DDoS detection script to execute.
- Example: 600 (10 minutes)

- Purpose: This timeout ensures that the Python script does not run indefinitely, which could indicate a problem. If the script takes longer than the specified timeout to complete, the Java application will terminate it and handle the situation accordingly.

## **6 - Blocklist CSV Path (BLOCKLIST\_CSV):**

- Definition: The file path where the blocklist CSV file is saved.
- Example: `"/home/sdn/ddos_blocklist.csv"`
- Purpose: The blocklist CSV file contains a list of IP addresses identified as part of a DDoS attack. The Java application reads this file to determine which IPs to block by installing flow rules on the ONOS controller.

### **6.4.2 Configuration Details**

Configuring the ONOS-based DDoS detection solution involves setting the above constants to match the specific environment in which the solution will be deployed. This configuration ensures that the Java application can effectively communicate with the ONOS controller, run the DDoS detection script, and manage the flow rules required to mitigate attacks.

#### **1 - Controller and Network Environment:**

- The `ONOS_URL` must be configured with the correct IP address and port number for the ONOS controller. This setting is critical because all REST API requests to ONOS depend on this URL.
- The network devices, such as switches and routers, must be properly configured to communicate with the ONOS controller. This configuration ensures that flow rules can be pushed effectively to these devices.

#### **2 - Security and Access Control:**

- The `USERNAME` and `PASSWORD` for ONOS must be set according to the credentials configured on the ONOS controller. These credentials are used for API authentication and must be kept secure to prevent unauthorized access.
- It is recommended to use secure methods for storing and handling these credentials, especially in production environments.

#### **3 - Script Execution:**

- The paths for `PYTHON_INTERPRETER` and `DDOS_DETECT_SCRIPT` must be accurate and point to valid files on the system. Incorrect paths will lead to script execution failures, which in turn will prevent the detection and mitigation of DDoS attacks.

- The `SCRIPT_TIMEOUT` should be set based on the expected duration for the DDoS detection script to run. If the script processes large amounts of data or performs complex computations, a longer timeout may be necessary. However, setting an excessively long timeout could delay response times.

#### 4 - Output Management:

- The `BLOCKLIST_CSV` path specifies where the Java application will write the blocklist file generated from the script's output. The application must have the necessary permissions to write to this location. Additionally, the path should be accessible to the Java application to read and apply block rules.

## 6.5 Core Functionality: Methods in the Application

This section delves into the core functionality of the Java application, highlighting the various methods implemented to automate the detection and mitigation of DDoS attacks in a Software-Defined Networking (SDN) environment. Each method plays a crucial role in ensuring that the system can efficiently detect potential threats and respond by dynamically modifying the network's behavior through the ONOS controller.

### 6.5.1 `main(String[] args)`

**Purpose:** The main method serves as the central orchestrator of the application, coordinating the entire DDoS detection and mitigation process.

**Functionality:**

- It begins by running a Python script designed to detect DDoS attack patterns within network traffic data.
- The output from the Python script is then used to generate a CSV file containing the IP addresses identified as sources of potential attacks.
- The method proceeds to read this blocklist and, based on the detected IP addresses, applies the necessary flow rules to the ONOS controller to block these malicious sources.

### 6.5.2 `runPythonScript(String command, int timeoutSeconds)`

**Purpose:** To execute the Python-based DDoS detection script and retrieve its output for further processing.

**Functionality:**

- The method constructs a command string that includes the path to the Python interpreter and the script.
- It executes this command in a separate process, capturing both standard output and error streams.
- The process is monitored to ensure it completes within a specified timeout. If the script takes too long, the process is forcibly terminated.
- The output from the script is returned as a string, which will later be used to create a blocklist of IP addresses.

### 6.5.3 createBlocklistCSV(String scriptOutput, String csvFilePath)

**Purpose:** To parse the output from the Python script and generate a CSV file listing the IP addresses involved in a potential DDoS attack.

**Functionality:**

- The method writes a header to the CSV file.
- It then processes the script's output line by line, extracting relevant IP addresses and their traffic counts.
- These extracted details are written into the CSV file, which is later used to apply block rules in the ONOS controller.

### 6.5.4 applyBlocklist(String csvFilePath)

**Purpose:** To read the blocklist CSV file and determine which IP address should be blocked by the ONOS controller.

**Functionality:**

- The method reads through the CSV file, skipping the header, and identifies the IP address with the highest traffic count, indicating it as the most likely source of a DDoS attack.
- It then calls the addBlockRuleToOnos method to apply a flow rule that blocks this IP address from sending traffic through the network.

### 6.5.5 `addBlockRuleToOnos(String srcIp)`

**Purpose:** To send a request to the ONOS controller to install a flow rule that blocks traffic from a specified source IP address.

**Functionality:**

- The method constructs a JSON payload that represents the flow rule, which includes details like the source IP to be blocked, the priority of the rule, and the device on which the rule should be applied.
- This payload is sent to the ONOS controller via an HTTP POST request.
- The response from the ONOS controller is checked to ensure the rule was successfully installed. If any errors occur, they are logged for troubleshooting.

### 6.5.6 `buildJsonPayload(String srcIp)`

**Purpose:** To create a JSON string that represents the flow rule for blocking a specific IP address.

**Functionality:**

- The method dynamically constructs a JSON object that includes the source IP, the device ID, and an empty treatment array (which implies that packets matching this rule will be dropped).
- The constructed JSON string is returned for use in the `addBlockRuleToOnos` method.

### 6.5.7 `fetchFromOnos(String endpoint)`

**Purpose:** To retrieve information from the ONOS controller, such as the current flow rules or device statuses.

**Functionality:**

- The method sends an HTTP GET request to the specified REST API endpoint on the ONOS controller.
- The response is read and returned as a string, which can be used to analyze the network state or debug issues.



### 6.5.8 encodeCredentials(String username, String password)

**Purpose:** To encode the ONOS REST API credentials for use in HTTP headers.

**Functionality:**

- The method concatenates the username and password, encodes them in Base64, and returns the encoded string.
- This string is then used in the Authorization header of all HTTP requests sent to the ONOS controller to ensure secure communication.

## 6.6 DDoS Attack Simulation

In order to test the solutions capability of detecting DDoS, the first step was to simulate a real world type DDoS attack. In the base solution, the DDoS simulation was done in a virtual environment using mininet. Mininet is a network emulator that creates a realistic virtual network for testing network protocols, configurations, and applications [14].

First was to setup a virtual network environment using mininet:

```
sudo mn --switch=ovsk,protocols=OpenFlow10 --mac --controller=remote,
ip=172.17.0.2,port=6653 --topo=linear,10 --link=tc,bw=10,delay=10ms
```

The command sets up a Mininet simulation with:

- A linear topology consisting of 10 Open vSwitch switches.
- Each switch uses the OpenFlow 1.0 protocol and connects to a remote controller located at IP 172.17.0.2 on port 6653.
- All hosts have MAC addresses that match their IP addresses.
- The links between switches have a bandwidth of 10 Mbps and a delay of 10 milliseconds.

To mimic a real world scenario the DDoS was simulated as following:

```
hping3 -S -c 800 -i u100 -p 80 10.0.0.1
```

This command is using hping3 which is a command-line tool used for crafting and sending custom TCP/IP packets[5] to simulate a SYN flood DDoS attack by sending 800 TCP SYN packets to the target IP address 10.0.0.1 on port 80 with a very short interval of 100 microseconds between packets. The attack is originating from a machine with IP address 10.0.0.2 (labeled as h2). This traffic is intended to be 80% of a DDoS attack simulation.

```
hping3 -S -c 200 -i u500000 -p 80 10.0.0.1
```

This command uses `hping3` to simulate normal network traffic from a machine (`h3`) with IP `10.0.0.3` to a target IP `10.0.0.1` on port `80` (commonly HTTP). It sends 200 SYN packets with a 0.5-second interval between packets, which is a slower rate, mimicking normal traffic behavior rather than an aggressive DDoS attack. This traffic is meant to represent 20% of the overall normal traffic in the simulation environment.

```
tcpdump -i h1-eth0 -w /home/sdn/output.pcap
```

This command uses `tcpdump` which is a powerful command-line packet analyzer used for capturing and analyzing network traffic to capture all network packets on the interface `h1-eth0` of the host `h1` (which has the IP address `10.0.0.1`). The captured packets are saved to a file located at `/home/sdn/output.pcap`. The output file will contain all the packets in PCAP format. Traffic capture is stopped after simulating the attack for 60 seconds.

## 6.7 Deployment and Testing Environment

This section outlines the deployment and testing environment utilized for implementing and validating our ONOS-based DDoS detection system.

### 6.7.1 Hardware and Virtualization Environment

The entire testing setup was conducted within a virtualized environment using VirtualBox version 7.0.18 r162988. The choice of a virtual environment allowed for easy configuration and reconfiguration of the network topology, as well as the isolation required for conducting controlled experiments.

**Virtual Machine Specifications:** Operating System: Ubuntu (configured with 8 GB of RAM, 100 GB of hard drive space, and 4 CPU cores). This setup provided sufficient resources for running multiple network simulations and processing traffic without causing significant slowdowns or performance bottlenecks.

### 6.7.2 Software Components

- **ONOS (Open Network Operating System):** software-defined networking (SDN) controller that provides centralized control over network devices. It manages flow rules

and network policies, making it a core component for the SDN-based DDoS detection solution.

- **Mininet:** Used for creating and managing the virtual network topology. Mininet allows for the simulation of a full network on a single machine, including switches, hosts, and links, which is ideal for testing P4-based applications.
- **Java Runtime Environment (JRE):** A runtime environment required to execute Java applications.
- **Hping:** This tool was utilized for generating network traffic, including both legitimate and attack traffic. Hping allows for the simulation of various types of DDoS attacks, which is essential for testing the system's detection capabilities under realistic conditions.
- **TCPdump:** Network protocol analyzers used for capturing and analyzing network traffic
- **Python 3.8:** The control plane application was developed in Python 3.8, leveraging several Python libraries to implement the machine learning model and manage communication with the P4 switch via P4Runtime.

## 7 P4-Based Solution

### 7.1 Development of P4 Application

This section outlines the development of the P4 application designed to implement DDoS detection and network traffic management on a programmable switch. The application leverages the P4\_16 programming language to process packets at the data plane level, allowing for highly efficient, low-latency handling of network packets directly within the switch. This P4 application is structured into several components, including packet parsing, ingress processing, ARP handling, packet forwarding, and a deparser for reassembling packets after processing. Each component is carefully designed to fulfill specific roles in the packet processing pipeline.

#### 7.1.1 Header Definitions

The application begins with defining the necessary header types: Ethernet, IPv4, and ARP. These headers represent the structure of the incoming network packets and are crucial for correctly parsing and processing the packets:

- **Ethernet Header:** Defines the source and destination MAC addresses (`srcAddr` and `dstAddr`), as well as the `etherType` field, which indicates the type of payload carried by the Ethernet frame (e.g., IPv4 or ARP).
- **IPv4 Header:** Defines various IPv4 header fields, including source (`srcAddr`) and destination (`dstAddr`) IP addresses, along with fields for version, header length (`ihl`), total length (`totalLen`), TTL (`ttl`), protocol, and header checksum (`hdrChecksum`).
- **ARP Header:** Specifies the fields required for handling ARP packets, including hardware type (`htype`), protocol type (`ptype`), hardware length (`hlen`), protocol length (`plen`), operation (`oper`), sender and target hardware addresses (`sha` and `tha`), and sender and target protocol addresses (`spa` and `tpa`).

### 7.1.2 Metadata and Header Structures

A metadata structure is defined to hold additional information needed during packet processing, such as source IP address (`src_ip`), destination IP address (`dst_ip`), and a flag (`ddos_detected`) to indicate if a DDoS attack is detected.

A headers struct is also defined to aggregate all the header types (`ethernet_t`, `ipv4_t`, and `arp_t`) into a single structure, which simplifies access and manipulation of packet headers during processing.

### 7.1.3 Packet Parser

The `MyParser` control block is responsible for extracting the headers from incoming packets. The parser operates through a series of states:

- **Start State:** Transitions to the `parse_ethernet` state, where the Ethernet header is extracted.
- **Parse Ethernet State:** Extracts the Ethernet header and transitions based on the `etherType` field. If the `etherType` indicates an IPv4 packet, it transitions to the `parse_ipv4` state. If it indicates an ARP packet, it transitions to the `parse_arp` state. Otherwise, it accepts the packet, ending the parsing process.
- **Parse IPv4 State:** Extracts the IPv4 header and stores the source and destination IP addresses in metadata. The state then transitions to accept the packet.
- **Parse ARP State:** Extracts the ARP header and then accepts the packet.

### 7.1.4 Ingress Processing

The `MyIngress` control block handles the core packet processing logic, ARP handling, and packet forwarding.

- Drop Action (drop): Drops the packet.
- Set Egress Port Action (set\_egress\_port): Sets the egress port (standard\_metadata.egress\_spec) for forwarding.
- ARP Response Action (send\_arp\_response): Constructs an ARP reply based on the request and sends it back to the requesting host.
- Table arp\_cache: Matches ARP requests based on the target protocol address (tpa) and either sends a response or does nothing (NoAction).
- Table forward: Matches packets based on the destination MAC address (dstAddr) and forwards them to the correct port or drops them by default.
- Apply Logic: Applies the appropriate table based on packet type: ARP packets trigger the arp\_cache table, while IPv4 packets trigger the forward table. All other packets are dropped by default.

#### 7.1.5 Egress and Checksum Controls

Egress Control (MyEgress): Currently, no operations are performed in this control block, but it serves as a placeholder for future egress processing requirements.

Checksum Computation: The MyComputeChecksum control block computes the IPv4 header checksum using a 16-bit one's complement sum. This ensures that any modifications to the header fields during processing are correctly accounted for, preserving the integrity of the packet.

#### 7.1.6 Deparser

The MyDeparser control block reassembles the packet headers in the correct order before the packet is transmitted out of the switch. It emits the Ethernet, IPv4, and ARP headers sequentially.

#### 7.1.7 Main Control Block

The V1Switch block instantiates and connects all the control blocks (MyParser, MyVerifyChecksum, MyIngress, MyEgress, MyComputeChecksum, and MyDeparser) to form the complete P4 program. This modular design allows for flexibility and scalability, enabling additional functionality to be integrated into the switch pipeline as needed.

## 7.2 Python Control Plane Application

The Python control plane application is designed to complement the P4 data plane program by managing network flows, handling packet-in events, and dynamically configuring the switch using P4Runtime APIs. This application operates as a critical component in detecting Distributed Denial of Service (DDoS) attacks and ensuring efficient traffic forwarding, leveraging both machine learning for anomaly detection and P4Runtime for flexible switch management. Below, we outline the key components and functionalities of the Python control plane application.

### 7.2.1 Setup and Configuration

The control plane application starts by importing necessary modules and configuring the environment. Key Python libraries such as `grpc`, `pandas`, `numpy`, and `joblib` are utilized. The application uses gRPC to communicate with the P4 switch and handle streaming of packet-in messages. The paths for the P4Info file, the P4 binary, and the pre-trained machine learning model are defined as constants at the beginning of the script:

```
P4INFO_FILE_PATH = "/build/ddos.p4info.txt"
P4_BINARY_PATH = "/build/ddos.json"
MODEL_PATH = "rf_model.sav"
file_path = "p4cap.csv"
```

The application also sets up logging to track operations and errors, which aids in debugging and monitoring the system:

```
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s %(message)s',
    handlers=[
        logging.FileHandler("controlplane.log"),
        logging.StreamHandler()
    ]
)
```

### 7.2.2 Machine Learning Model Integration

The application loads a pre-trained Random Forest model using `joblib`. This model is employed to classify network traffic and detect potential Distributed Denial of Service (DDoS) attacks:

```
model = joblib.load(MODEL_PATH)
```

### 7.2.3 Data Loading and Preparation

**Data Loading and Preprocessing:** The `load_and_prepare_data` function reads the CSV file containing network traffic data, selects relevant features, and preprocesses the data for the machine learning model:

- **Feature Selection:** It selects features such as 'Flow Bytes/s', 'Flow Packets/s', and various inter-arrival times to feed into the model.
- **Data Cleaning:** It handles infinite values, replaces them with NaN, and then fills NaNs with the mean of the respective columns to ensure data integrity for model prediction:

```
ntraffic_data = pd.read_csv(file_path)
X_new = ntraffic_data[features].values
X_new = pd.DataFrame(X_new).replace([np.inf, -np.inf], np.nan).fillna(X_new.mean())
```

### 7.2.4 Switch Connection and Pipeline Configuration

**Switch Connection Setup:** The `connect_to_switch` function establishes a connection to the P4 switch using P4Runtime, configures the switch's forwarding pipeline, and ensures that the control plane has master arbitration:

```
switch_connection = SwitchConnection(
    name='switch',
    address=p4runtime_address,
    device_id=device_id,
    proto_dump_file='logs/p4runtime_requests.txt'
)
switch_connection.MasterArbitrationUpdate()
switch_connection.SetForwardingPipelineConfig(
    p4info=p4info_helper.p4info,
    bmv2_json_file_path=p4_binary_path
)
```

### 7.2.5 Table Management and ARP Handling

**Default Forwarding Rules:** The `add_default_forwarding_rules` function installs forwarding rules into the switch to route traffic based on the destination MAC address:

```

table_entry_1 = p4info_helper.buildTableEntry(
    table_name="MyIngress.forward",
    match_fields={"hdr.ethernet.dstAddr": "00:00:00:00:01:01"},
    action_name="MyIngress.set_egress_port",
    action_params={"port": 1}
)
switch_connection.WriteTableEntry(table_entry_1)

```

ARP Entries: The `add_arp_entry` function adds entries to the ARP cache table, allowing the switch to handle ARP requests by responding with the correct MAC and IP address mappings:

```

table_entry = p4info_helper.buildTableEntry(
    table_name="MyIngress.arp_cache",
    match_fields={"hdr.arp.tpa": ip_addr},
    action_name="MyIngress.send_arp_response",
    action_params={"src_mac": mac_addr, "dst_mac": "ff:ff:ff:ff:ff:ff", "src_ip": ip}
)
switch_connection.WriteTableEntry(table_entry)

```

### 7.2.6 DDoS Detection and ARP Entry Management

**DDoS Detection and ARP Removal:** The `analyze_traffic` function uses the pre-trained model to analyze traffic and detect potential DDoS attacks:

- **Model Prediction:** The model classifies the traffic and identifies DDoS traffic based on preprocessed features.
- **ARP Entry Removal:** If a source IP is identified as a potential DDoS attacker, its corresponding ARP entry is removed to mitigate the attack.

```

if not source_counts.empty:
    ip_to_block = source_counts.idxmax()
    ip_int = ip_to_int(ip_to_block)
    remove_arp_entry(switch_connection, p4info_helper, ip_int)

```

### 7.2.7 Main Function and Application Execution

**Main Function:** The main function orchestrates the initialization and execution of the control plane application. It sets up the switch connection, installs initial rules, and continuously



analyzes traffic for DDoS detection:

```
switch_connection, p4info_helper = connect_to_switch(
    p4info_file_path, p4_binary_path, p4runtime_address, device_id
)
add_default_forwarding_rules(switch_connection, p4info_helper)
analyze_traffic(switch_connection, p4info_helper, file_path)
```

### 7.3 Shutdown and Cleanup

The application ensures that all switch connections are properly shut down on exit, preventing any potential resource leaks or dangling connections:

```
finally:    ShutdownAllSwitchConnections()
```

### 7.4 System Architecture and Data Flow

This section provides an in-depth overview of the system architecture and data flow of our programmable network solution designed for dynamic DDoS detection and mitigation. The architecture integrates a P4-programmable switch, a Python-based control plane application, and a machine learning model.

#### 7.4.1 Introduction to System Architecture

The primary objective of the proposed system architecture is to enhance network security by detecting and mitigating DDoS attacks in real-time while maintaining efficient traffic management. The architecture consists of three main components: a P4-programmable switch that processes packets directly in the data plane, a Python control plane application that manages the switch's behavior via P4Runtime, and a machine learning model that provides intelligent DDoS detection based on network flow features.

#### 7.4.2 Overview of Components

**1 - P4 Programmable Data Plane:** The data plane is implemented using the P4 language and runs on a P4-capable switch. It includes:

- **Packet Parsing:** Extracts Ethernet, IPv4, and ARP headers from incoming packets.
- **Ingress Processing:** Contains logic to drop malicious packets, forward legitimate packets, and respond to ARP requests.

- **Table Management:** Includes tables for ARP caching and forwarding, allowing dynamic modification based on control plane instructions.

**2- Python Control Plane Application:** The control plane is implemented in Python and communicates with the P4 switch using P4Runtime. Key components include:

- **Machine Learning Model:** Uses a pre-trained Random Forest model to detect anomalies in network traffic.
- **Traffic Analyzer:** Loads traffic data, processes it, and uses the model to predict potential DDoS attacks.
- **P4Runtime Controller:** Manages table entries in the data plane, adding or removing rules dynamically based on the analysis.

**3 - P4Runtime API:** Acts as a bridge between the data plane and control plane, allowing the Python application to read and modify the switch's forwarding tables.

#### 7.4.3 System Workflow and Data Flow

The system operates through a series of well-defined steps, from packet ingress to final forwarding or dropping, based on real-time traffic analysis.

**1 - Packet Arrival:** Network packets arrive at the P4 switch. The parser extracts headers (Ethernet, IPv4, ARP) to identify packet types.

##### **2 - Ingress Processing:**

- If the packet is an ARP request, the switch checks the arp\_cache table to determine if a response should be sent. If no entry is found, the packet is dropped.
- If the packet is IPv4, the forward table determines the egress port based on the destination MAC address. If no match is found, the packet is dropped.

##### **3 - Control Plane Decision Making:**

- The control plane periodically analyzes traffic using the analyze\_traffic function, applying the machine learning model to detect potential DDoS attacks based on flow statistics extracted from packet headers.
- Upon detecting an attack, the control plane issues commands via P4Runtime to modify the data plane behavior. This can include removing ARP entries to block traffic from suspicious IPs or adding forwarding rules to reroute traffic.

**4 - Dynamic Table Updates:** The control plane uses P4Runtime to update switch tables, adding or removing rules based on real-time traffic analysis. This ensures the data plane responds dynamically to potential threats.

#### 7.4.4 Data Flow Diagram

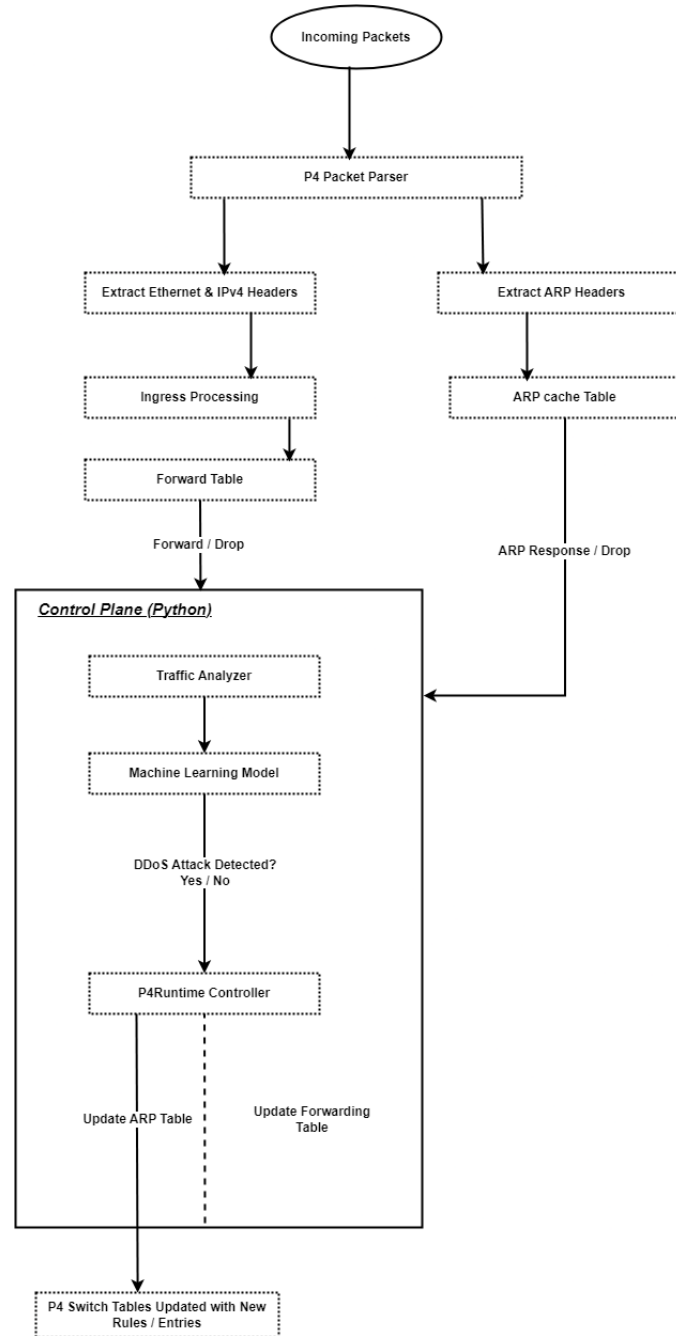


Figure 9: Data Flow

#### 7.4.5 Scalability and Flexibility of the Architecture

1. **Scalability:** The architecture is designed to be scalable by leveraging the programmable nature of P4 switches and the dynamic capabilities of the control plane. As network traffic grows, the control plane can adapt by deploying more sophisticated machine learning models or by expanding its traffic analysis capabilities.

2. **Flexibility:** The system is highly flexible due to its programmable nature. The control plane can dynamically adjust rules based on real-time traffic patterns and can be updated to incorporate new security policies or detection mechanisms without requiring changes to the physical hardware.

#### 7.4.6 Challenges and Considerations

- **Latency:** Introducing a control plane that dynamically modifies forwarding rules can introduce latency, especially if decisions depend on complex machine learning models. Optimization and careful selection of features for the model are essential to minimize processing time.
- **Synchronization:** Maintaining synchronized state between the control plane and data plane is crucial to ensure consistent network behavior. Any lag or discrepancy can lead to packets being dropped or forwarded incorrectly.
- **Scalability Limitations:** While the architecture is scalable, the control plane's ability to process and respond to high volumes of traffic may be limited by the computational resources available for running machine learning models and handling gRPC communications.
- **Security and Reliability:** Ensuring that the control plane itself is secure and reliable is paramount. If compromised, an attacker could manipulate network behavior, potentially exacerbating a DDoS attack or causing other disruptions.

### 7.5 Deployment and Testing Environment

This section outlines the deployment and testing environment utilized for implementing and validating our P4-based DDoS detection system. The environment was carefully configured to simulate real-world scenarios, allowing for comprehensive testing of our system's capabilities in detecting and mitigating network attacks effectively.

#### 7.5.1 Hardware and Virtualization Environment

The entire testing setup was conducted within a virtualized environment using VirtualBox version 7.0.18 r162988. The choice of a virtual environment allowed for easy configuration and reconfiguration of the network topology, as well as the isolation required for conducting controlled experiments.

**Virtual Machine Specifications:** Operating System: Ubuntu (configured with 8 GB of RAM, 100 GB of hard drive space, and 4 CPU cores). This setup provided sufficient resources for running multiple network simulations and processing traffic without causing

significant slowdowns or performance bottlenecks.

### 7.5.2 Software Components

The deployment relied on several key software components, each playing a crucial role in the setup and testing of the system:

- **Mininet:** Used for creating and managing the virtual network topology. Mininet allows for the simulation of a full network on a single machine, including switches, hosts, and links, which is ideal for testing P4-based applications.
- **BMv2 (Behavioral Model version 2):** This is a software switch that supports the execution of P4 programs. BMv2 was configured to operate as a P4-programmable switch in our environment, providing the necessary functionality to parse, process, and forward packets as dictated by the P4 program.
- **P4 Compiler (P4c):** The P4c compiler was used to compile the P4 code into a format that can be executed by BMv2. This step is critical for ensuring that the switch correctly interprets and implements the specified forwarding rules and DDoS detection logic.
- **Hping:** This tool was utilized for generating network traffic, including both legitimate and attack traffic. Hping allows for the simulation of various types of DDoS attacks, which is essential for testing the system's detection capabilities under realistic conditions.
- **TCPdump:** Network protocol analyzers used for capturing and analyzing network traffic
- **Python 3.8:** The control plane application was developed in Python 3.8, leveraging several Python libraries to implement the machine learning model and manage communication with the P4 switch via P4Runtime.

### 7.5.3 Machine Learning and Control Plane Libraries

Several Python libraries were employed to support the control plane's operations and the integration of machine learning models:

- **gRPC:** Used to establish communication between the Python control plane application and the P4 switch. gRPC enables the control plane to send and receive messages from the switch, such as packet-in notifications and table updates.

- **Pandas, NumPy, and Joblib:** These libraries were used for data manipulation, numerical computations, and loading the machine learning model. Pandas was particularly useful for handling flow statistics data, while NumPy provided the necessary tools for performing mathematical operations on arrays of data.
- **Scikit-Learn:** This library was used to develop and train the Random Forest model used for DDoS detection. The trained model was then integrated into the control plane application to make real-time predictions based on incoming network traffic data.

#### 7.5.4 Network Topology Configuration

The network topology was created using Mininet, and it consisted of a single P4-programmable switch (s1) and four host machines (h1, h2, h3, and h4). Each host was configured with a unique IP address and MAC address, simulating different network entities. The switch (s1) was connected to a gRPC server on port 50051 to facilitate communication with the Python control plane.

##### Host Configuration:

- h1: IP 10.0.1.1, MAC 00:00:00:00:01:01
- h2: IP 10.0.1.2, MAC 00:00:00:00:01:02
- h3: IP 10.0.1.3, MAC 00:00:00:00:01:03
- h4: IP 10.0.1.4, MAC 00:00:00:00:01:04

#### 7.5.5 Testing Procedures

The testing environment was designed to replicate a variety of network conditions to evaluate the system's robustness and effectiveness in DDoS detection:

- **Traffic Generation:** Hping was employed to generate both normal and malicious traffic. Different types of DDoS attacks, such as SYN floods and UDP floods, were simulated to test the system's ability to detect and mitigate these attacks in real-time.
- **Packet Capture and Analysis:** BMv2 allowed for the capture of traffic at various stages of processing, providing insights into how packets were handled by the P4 switch. Packet captures were analyzed using tcpdump to verify that packets were correctly forwarded or dropped based on the control plane's instructions.
- **Control Plane Interaction:** The Python control plane application dynamically managed the switch's behavior using P4Runtime, updating table entries and forwarding rules based on real-time analysis of network traffic.

### 7.5.6 Environment Logs and Monitoring

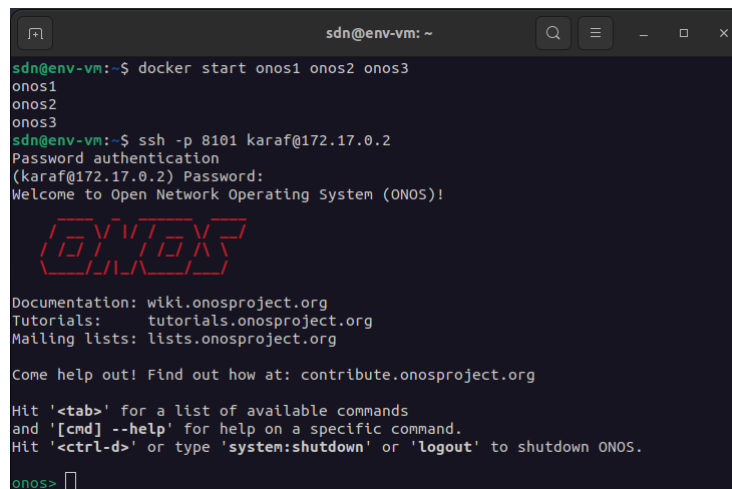
To ensure comprehensive testing and debugging capabilities, logs were maintained for all components:

- **Switch Logs:** Captured all activities within the BMv2 switch, including packet processing and rule application, aiding in the verification of the P4 program's functionality.
- **Control Plane Logs:** Provided detailed insights into the decision-making processes of the Python control plane, including machine learning model predictions and P4Runtime interactions.
- **Packet Traces:** Generated by capturing traffic with tcpdump, these traces were used to analyze the precise behavior of the network.

## 8 Results

### 8.1 DDOS Detection Results from ONOS solution

Figure 10 shows that the simulation begins by launching the ONOS cluster and establishing an SSH connection to it.



```

sdn@env-vm: ~
sdn@env-vm:~$ docker start onos1 onos2 onos3
onos1
onos2
onos3
sdn@env-vm:~$ ssh -p 8101 karaf@172.17.0.2
Password authentication
(karaf@172.17.0.2) Password:
Welcome to Open Network Operating System (ONOS)!

  ONOS

Documentation: wiki.onosproject.org
Tutorials:    tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown ONOS.

onos>

```

Figure 10: Starting Docker

Next, as depicted in 11, the ONOS cluster is accessed via a web browser by navigating to the ONOS URL, which consists of the ONOS IP address. Upon arrival at the login prompt, users enter their ONOS credentials. After successful authentication, the topology section is accessible through the menu bar on the left. The next step in the simulation involves creating a Mininet topology using the following command:

```
sudo mn --switch=ovsk,protocols=OpenFlow10 --mac --controller=remote,
ip=172.17.0.2,port=6653 --topo=linear,10 --link=tc,bw=10,delay=10ms
```

Once the topology is successfully established, its functionality is verified by executing a ping from host h2 to host h1, ensuring network connectivity and that the setup operates as expected.

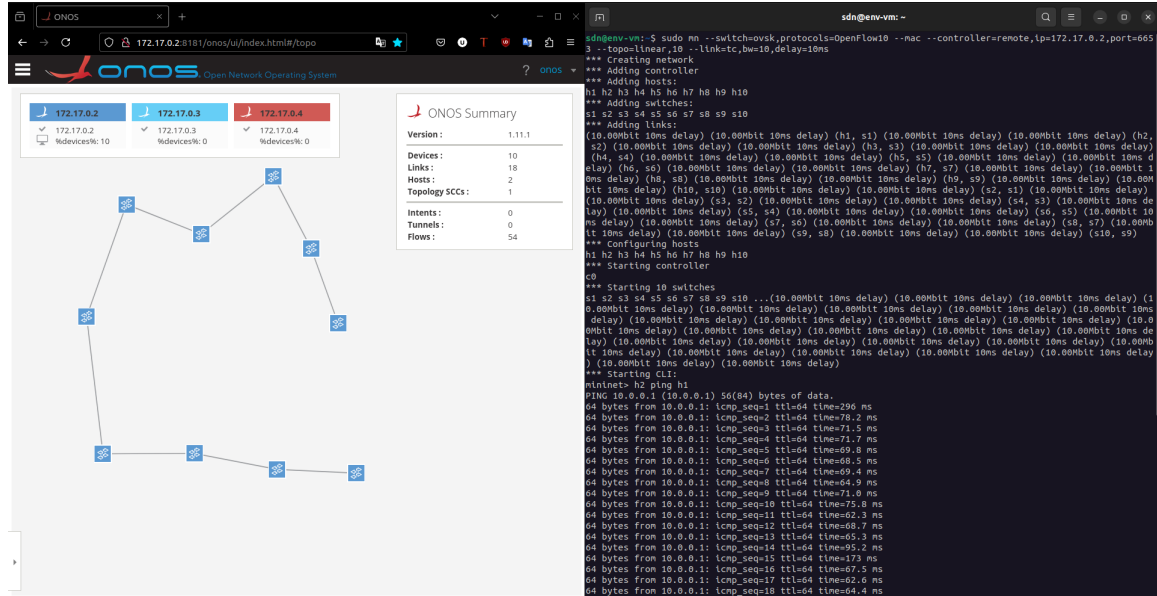


Figure 11: Docker Visualisation and Mininet Start

After the Initial test, the java application is started to perform the ddos detection and add flow rules to block the malicious IP. In figure 12, it can be seen that a rule is added to block the IP 10.0.0.2 as it was detected to be the IP that had the most count which essentially means that it was the DDoS source. This detection is done on the data that was gathered from performing a DDoS attack simulation where 10.0.0.2 was the DDoS source.



```

Name: count, dtype: int64
Source IPs and their counts for DDoS traffic:
Source IP
10.0.0.2      697
10.0.0.3      142
8.6.0.1        1
136.204.2.7   1
137.66.2.7    1
Name: count, dtype: int64

Blocklist CSV created at /home/sdn/ddos_blocklist.csv
JSON Payload: {
  "flows": [
    {
      "priority": 40000,
      "timeout": 0,
      "isPermanent": true,
      "deviceId": "of:0000000000000001",
      "treatment": {
        "instructions": []
      },
      "selector": {
        "criteria": [
          {
            "type": "IPV4_SRC",
            "ip": "10.0.0.2/32"
          }
        ]
      }
    }
  ]
}

Blocked IP with highest count: 10.0.0.2

Process finished with exit code 0

```

Figure 12: Adding Blocking rule after DDoS Detection

After adding the blocking rule, its effectiveness is tested by returning to Mininet and attempting to ping from host h2 (IP 10.0.0.2). In figure 13, the green ping result shows successful communication from h2 to h1 before the rule was applied. In contrast, the red ping result, recorded immediately after the rule's implementation, shows that all packets were dropped, confirming the rule's successful operation in mitigating the DDoS source.

```

mininet> h2 ping h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data:
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=195 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=68.6 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=63.2 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=73.2 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=62.4 ms
64 bytes from 10.0.0.1: icmp_seq=6 ttl=64 time=115 ms
^C
--- 10.0.0.1 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5013ms
rtt min/avg/max/mdev = 62.384/96.222/195.234/47.736 ms
mininet> h2 ping h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
^C
--- 10.0.0.1 ping statistics ---
27 packets transmitted, 0 received, 100% packet loss, time 26610ms
mininet>

```

Figure 13: Ping Test

## 8.2 DDOS Detection Results from p4 solution

The initial step in simulating the P4 solution involves launching the P4 application. This is achieved by executing a make command, which runs a makefile. The makefile initiates

Mininet, constructs the network topology from a JSON file, and starts the BMv2 software switch. This process is illustrated in Figure 14. Upon successful execution, the simulation transitions to the Mininet CLI.

```

Reading topology file.
Building mininet topology.
Switch port mapping:
s1:
1:h1
2:h2
3:h3
4:h4
s1 -> gRPC port: 50051
*****
h1
default interface: h1-eth0      10.0.1.1      00:00:00:00:01:01
*****
h2
default interface: h2-eth0      10.0.1.2      00:00:00:00:01:02
*****
h3
default interface: h3-eth0      10.0.1.3      00:00:00:00:01:03
*****
h4
default interface: h4-eth0      10.0.1.4      00:00:00:00:01:04
*****
Starting mininet CLI

=====
Welcome to the BMV2 Mininet CLI!
=====
Your P4 program is installed into the BMV2 software switch
and your initial runtime configuration is loaded. You can interact
with the network using the mininet CLI below.

To inspect or change the switch configuration, connect to
its CLI from your host operating system using this command:
    simple_switch_CLI --thrift-port <switch thrift port>

To view a switch log, run this command from your host OS:
    tail -f /home/p4/tutorials/exercises/SwitchTree/logs/<switchname>.log

To view the switch output pcap, check the pcap files in /home/p4/tutorials/exercises/SwitchTree/pcaps:
for example run:  sudo tcpdump -xxx -r s1-eth1.pcap

To view the P4Runtime requests sent to the switch, check the
corresponding txt file in /home/p4/tutorials/exercises/SwitchTree/logs:
for example run:  cat /home/p4/tutorials/exercises/SwitchTree/logs/s1-p4runtime-requests.txt

mininet>

```

Figure 14: Starting P4 app

After launching the P4 application, the next step is to initialize the control plane. The control plane application configures forwarding rules, assigns MAC addresses to IPs, and executes the `ddos_detect.py` script, which utilizes a trained machine learning model to predict potential DDoS attacks based on the available data. Following these predictions, the ARP entry for the malicious IP h2 (10.0.1.2) is removed from the rules table. This process is illustrated in Figure 15.

```

2024-08-31 13:26:05,856 Added default forwarding rule for 00:00:00:00:01:01 -> port 1
2024-08-31 13:26:05,856 Added default forwarding rule for 00:00:00:00:01:02 -> port 2
2024-08-31 13:26:05,859 Added default forwarding rule for 00:00:00:00:01:03 -> port 3
2024-08-31 13:26:05,860 Added ARP entry for 10.0.1.1 -> 00:00:00:00:01:01
2024-08-31 13:26:05,861 Added ARP entry for 10.0.1.2 -> 00:00:00:00:01:02
2024-08-31 13:26:05,861 Added ARP entry for 10.0.1.3 -> 00:00:00:00:01:03
2024-08-31 13:26:05,862 Added ARP entry for 10.0.1.4 -> 00:00:00:00:01:04
Predictions
0      1
1      1
2      1
3      0
4      0
...    ...
1061    0
1062    0
1063    0
1064    0
1065    0

[1066 rows x 1 columns]
Class distribution in the new data:
Predictions
0      855
1      211
Name: count, dtype: int64
Source IPs and their counts for DDoS traffic:
Source IP
10.0.1.2      206
8.6.0.1        3
10.0.1.3        2
Name: count, dtype: int64
2024-08-31 13:26:05,908 Removed ARP entry for IP: 167772418
2024-08-31 13:26:05,908 Removed ARP entry for IP with highest DDoS count: 10.0.1.2

```

Figure 15: Control Plane Start and Logs

After the DDoS detection process is complete and the malicious IP is blocked, the next step is to verify that the system functions as intended. To confirm the success of the DDoS detection, we attempt to ping from the blocked IP address. As shown in Figure 16, the ping from the blocked IP fails, indicating that the blocking mechanism is working correctly.

```

mininet> h2 ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
^C
--- 10.0.1.1 ping statistics ---
9 packets transmitted, 0 received, 100% packet loss, time 8029ms

```

Figure 16: Pinging from blocked IP after ddos detection

To ensure that the blocking mechanism did not interfere with other hosts, ping from h3 to 10.0.1.1 is initiated. The successful ping, as expected, confirms that only the malicious IP was blocked while all other hosts remain unaffected.

```

mininet> h3 ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=3.13 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=1.03 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.715 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=64 time=0.707 ms
64 bytes from 10.0.1.1: icmp_seq=5 ttl=64 time=1.01 ms
64 bytes from 10.0.1.1: icmp_seq=6 ttl=64 time=0.855 ms
64 bytes from 10.0.1.1: icmp_seq=7 ttl=64 time=0.647 ms
64 bytes from 10.0.1.1: icmp_seq=8 ttl=64 time=0.529 ms
64 bytes from 10.0.1.1: icmp_seq=9 ttl=64 time=0.396 ms
64 bytes from 10.0.1.1: icmp_seq=10 ttl=64 time=0.641 ms
64 bytes from 10.0.1.1: icmp_seq=11 ttl=64 time=1.96 ms
64 bytes from 10.0.1.1: icmp_seq=12 ttl=64 time=1.04 ms
64 bytes from 10.0.1.1: icmp_seq=13 ttl=64 time=0.798 ms
64 bytes from 10.0.1.1: icmp_seq=14 ttl=64 time=1.05 ms
64 bytes from 10.0.1.1: icmp_seq=15 ttl=64 time=1.13 ms
64 bytes from 10.0.1.1: icmp_seq=16 ttl=64 time=1.71 ms
64 bytes from 10.0.1.1: icmp_seq=17 ttl=64 time=0.923 ms
64 bytes from 10.0.1.1: icmp_seq=18 ttl=64 time=0.922 ms
64 bytes from 10.0.1.1: icmp_seq=19 ttl=64 time=1.00 ms
64 bytes from 10.0.1.1: icmp_seq=20 ttl=64 time=0.953 ms
64 bytes from 10.0.1.1: icmp_seq=21 ttl=64 time=0.975 ms
64 bytes from 10.0.1.1: icmp_seq=22 ttl=64 time=0.952 ms
64 bytes from 10.0.1.1: icmp_seq=23 ttl=64 time=1.12 ms

```

Figure 17: pinging from non blocked IP

## 9 Discussion

### 9.1 Achieved Results Interpretation

#### 9.1.1 Detection Accuracy and Granularity

**1 - ONOS Solution:** The ONOS solution requires manual initiation of the Java application to run the detection script and apply flow rules. This introduces a delay between the detection of a DDoS attack and the application of mitigation measures. As a result, there may be a time gap during which the network remains vulnerable to ongoing attacks until the Java application is run.

**2 - P4 Solution:** The P4 solution offers immediate detection and mitigation upon control plane startup. As the DDoS detection is integrated directly into the control plane logic, the system can rapidly detect and respond to attacks without requiring separate execution steps. This integration ensures minimal delay between detection and mitigation.

#### 9.1.2 Operational Efficiency

**1 - ONOS Solution:** The efficiency of the ONOS solution depends on the timely execution of the Java application. While it provides flexibility in managing and updating flow rules, it requires external management to initiate detection and apply rules, which could be less efficient in scenarios requiring rapid response.

**2 - P4 Solution:** The P4 solution is more operationally efficient in environments where immediate threat detection and response are crucial. Since detection and mitigation are part of the control plane's core functions, there is no need for additional applications to run separately, leading to faster and more streamlined operations.

#### 9.1.3 Flexibility and Control

**1 - ONOS Solution:** The ONOS solution provides greater flexibility in how detection scripts and mitigation actions are managed. Network administrators can easily update the detection logic or mitigation strategies by modifying the Java application or detection scripts. This makes it more adaptable to different network environments and evolving threats.

**2 - P4 Solution:** The P4 solution, while faster, is more rigid as it embeds detection and mitigation logic directly within the control plane application. Changes to detection algorithms or mitigation rules may require reprogramming the control plane logic or P4 switch configurations, making it less flexible in adapting to new types of attacks.

#### 9.1.4 Scalability and Maintenance

The P4 solution is more scalable compared to the ONOS solution because it distributes the processing load across multiple switches, reducing dependency on a central controller and avoiding the bottlenecks associated with centralized control. This allows the network to handle larger scales of traffic and more switches while maintaining low latency and high performance, making it ideal for dynamic, high-speed network environments where real-time responses are crucial. However, both solutions have their limitations in scalability, and the choice between them depends on the specific requirements of the network environment.

#### 9.1.5 Overall Effectiveness

**1 - ONOS Solution:** Effective for networks where flexibility and ease of integration with different detection methods are prioritized. It offers control over when and how detection scripts are run and flow rules are applied, which can be advantageous in certain contexts.

**2 - P4 Solution:** Superior for environments requiring immediate, real-time DDoS detection and mitigation. Its integration within the control plane allows for rapid response, making it highly effective in scenarios where latency and quick adaptation to threats are critical.

### 9.2 Implementation of the P4 Solution in a Real-World Scenario

To effectively implement the P4-based DDoS detection solution in real-world scenarios, several factors must be considered to leverage its advantages and mitigate potential drawbacks. This section explores how the solution can be deployed in a real-world network environment, the benefits it offers, and the limitations it may encounter. We also evaluate whether this solution addresses the research question regarding the enhancement of real-time DDoS detection and mitigation through the integration of Software-Defined Networking (SDN), programmable data planes, and machine learning.

#### 9.2.1 Deployment in Wide-Area Networks (WANs)

- The P4-based solution can be deployed in WANs by integrating programmable switches that support the P4 language. These switches would be strategically placed at key network points, such as ingress and egress points, to monitor and control incoming and outgoing traffic.
- The Python control plane application, which interfaces with these P4 switches via P4Runtime, would be hosted on dedicated servers within the network. This setup allows the control plane to maintain a real-time connection with multiple switches, manage flow tables, and execute machine learning models to detect and respond to DDoS attacks.

### 9.2.2 Integration with Existing Network Infrastructure

- The solution can be integrated into existing network infrastructures that already utilize SDN controllers. The P4-programmable switches can replace or work alongside traditional switches, providing enhanced functionality without requiring a complete overhaul of the network.
- The control plane application can be integrated with existing network management systems to ensure a seamless transition and reduce the operational overhead associated with maintaining separate systems for traditional and P4-enabled devices.

### 9.2.3 Dynamic Traffic Management and DDoS Mitigation

- The P4 switches enable real-time packet inspection and forwarding decisions based on rules dynamically pushed by the control plane. This capability allows for immediate responses to detected DDoS attacks, such as dropping malicious traffic or rerouting legitimate traffic to minimize impact.
- The Python control plane application leverages a machine learning model to analyze traffic patterns and detect anomalies indicative of DDoS attacks. The model's predictions are used to update the P4 switch's flow tables, blocking malicious IPs and mitigating the attack at the data plane level.

### 9.2.4 Advantages

#### 1 - Real-Time Detection and Response:

- The integration of P4-programmable switches enables real-time processing and decision-making directly in the data plane. This reduces latency and allows for rapid detection and mitigation of DDoS attacks, ensuring a timely response to threats.
- By using machine learning models in the control plane, the solution can adapt to evolving attack patterns and refine its detection capabilities over time, enhancing the accuracy and reliability of DDoS detection.

#### 2 - High Scalability and Flexibility:

- The solution's architecture is highly scalable due to its modular design. Additional P4 switches and control plane instances can be deployed as network demands grow, allowing the solution to handle increased traffic volumes without a significant performance hit.
- The programmable nature of P4 allows for flexibility in defining and updating packet processing rules. This means the solution can easily adapt to new types of attacks or changes in network policy without requiring hardware changes.

### 9.2.5 Drawbacks and Challenges

#### 1 - Complexity of Integration:

- Integrating P4-programmable switches and a Python-based control plane with existing network infrastructure can be complex. It is important to ensure that the existing system configuration will be compatible with this solution to prevent disruptions to network services.
- The control plane needs to manage multiple P4 switches efficiently, which can become challenging as the network scales. Ensuring consistent updates and maintaining synchronized states between the control plane and data plane devices is critical.

#### 2 - Resource Requirements:

- Running machine learning models and managing real-time traffic analysis requires significant computational resources. In high-traffic environments, this could lead to increased operational costs and potential performance bottlenecks if not properly managed.

#### 3 - Latency and Processing Overhead:

- While the solution aims to minimize latency, the introduction of a control plane that dynamically updates flow rules can introduce some delay, especially if machine learning models are computationally intensive.
- In scenarios where rapid response is crucial, the time taken to analyze traffic and update the data plane must be minimized to avoid any noticeable impact on network performance.

## 9.3 Does the Solution Answer the Research Question?

The research question explores how the integration of SDN, programmable data planes, and machine learning can enhance real-time detection and mitigation of sophisticated DDoS attacks in wide-area networks while overcoming the challenges of seamless integration, timely response, and system scalability.

The P4-based solution addresses the research question in several ways:

1. **Seamless Integration:** By utilizing P4-programmable switches and SDN principles, the solution integrates with existing network infrastructures, enabling a seamless deployment without requiring extensive changes to the current setup. The control plane's use of standardized protocols ensures compatibility and smooth operation across different network components.

2. **Timely Response:** The solution's ability to perform real-time packet processing at the data plane level, combined with machine learning-driven analysis in the control plane, ensures a rapid response to DDoS attacks. The system is designed to detect threats as they emerge and immediately apply mitigation measures, reducing the window of vulnerability.
3. **Scalability:** The modular design of the solution allows for easy scaling across large networks. Additional P4 switches and control plane applications can be deployed to handle increased traffic and provide broader coverage, demonstrating the solution's scalability and adaptability to various network sizes and configurations.

### 9.3.1 Implications for Network Security

The proposed approach has significant implications for network security, particularly in enhancing the ability to detect and mitigate DDoS attacks in real-time:

- **Enhanced Detection Capabilities:** By leveraging programmable switches and integrating machine learning, the system offers advanced detection capabilities that can adapt to new and sophisticated attack patterns, providing a more proactive security posture.
- **Improved Resilience:** The ability to dynamically update network rules in response to detected threats enhances the resilience of the network, allowing it to maintain performance and availability even under attack conditions.
- **Foundation for Future Security Applications:** The architecture provides a flexible foundation that can be extended to other network security applications, such as intrusion detection, anomaly detection, and automated response mechanisms, thereby offering a comprehensive approach to network security.



### 9.4 Comparison with Existing Solutions

Category	3D-PM [10]	DataPlane-ML [18]	Proposed P4 Solution
Scalability	Limited to specific attack types; may struggle in large networks	Moderate; better scalability but limited by data plane resources	High scalability across wide-area networks
Seamless Integration	Minimal integration with SDN controller; mostly data plane-focused	Limited controller involvement; relies on some configurations	Full integration with SDN controllers for dynamic updates
Response Time	Fast detection at data plane level	Low latency due to data plane processing	low latency with adaptive mechanisms
Resource Efficiency	Efficient for specific scenarios; less adaptable to changes	High resource use at data plane; potential performance impacts	Optimized for low resource use; maintains performance under load
Detection Accuracy	High accuracy for targeted attack types	High accuracy for various attack types; can impact resources	Broad accuracy across multiple attack types
Flexibility	Focused on specific DDoS attack patterns	More flexible than 3D-PM; limited by complexity	Highly flexible; adaptable to different conditions and attack types

Table 6: Comparison of DDoS Detection Solutions in SDN Environments

### 9.5 Potential Improvements and Future Work

While the current system demonstrates significant benefits, there are several areas for potential improvement and future work:

- Enhanced Model Training:** Future work could focus on continuously retraining the machine learning model with new data to ensure it remains effective against evolving

threats. Incorporating unsupervised learning techniques could also help in identifying previously unseen attack patterns.

- **Optimizing Resource Management:** Investigating methods to optimize the computational resources required by the control plane, such as using lightweight models or offloading some processing tasks to specialized hardware (e.g., GPUs), could enhance the system's performance in high-traffic environments.
- **Improved Rule Management Algorithms:** Developing more sophisticated algorithms for managing dynamic rule updates could help ensure consistency and minimize the risk of conflicts or redundant rules, especially in large and complex networks.
- **Field Deployment and Real-World Testing:** Future work should include deploying the system in a real-world environment to evaluate its performance under realistic conditions and gather feedback for further refinement.

## 10 Conclusion

This research has explored the integration of Software-Defined Networking (SDN), programmable data planes, and machine learning (ML) to enhance the detection and mitigation of Distributed Denial of Service (DDoS) attacks in modern network infrastructures. The research has focused on developing two distinct solutions: an ONOS-based SDN approach using a Java application for DDoS detection and a P4-based solution leveraging a programmable switch integrated with a Python control plane and machine learning model. The ONOS-based solution demonstrated flexibility and adaptability in managing detection scripts and flow rules. It allowed for dynamic updates and the integration of various detection algorithms, providing a robust framework suitable for environments where rapid adaptation to evolving network conditions is necessary. However, this approach showed limitations in scenarios requiring immediate threat detection due to its reliance on periodic execution of the Java application to detect and mitigate threats. This introduces a potential delay between the detection of an attack and the application of mitigation measures, which could leave the network vulnerable during this period. On the other hand, the P4-based solution embedded DDoS detection capabilities directly into the data plane, enabling a low-latency, real-time response to network threats. This solution integrated a machine learning model within the network's control logic, allowing for immediate detection and response without the need for separate applications. The results from simulations using Mininet highlighted the superior performance of the P4-based approach in scenarios requiring immediate threat detection and mitigation. The programmable nature of P4 allowed for fine-grained control over packet processing and offered the flexibility to implement custom and dynamic security measures directly within the network infrastructure. Both solutions were evaluated through

extensive simulations designed to replicate real-world network environments. The performance of each approach was analyzed based on detection accuracy, operational efficiency, scalability, and flexibility. The findings indicate that while the ONOS-based solution offers greater adaptability and control over detection and mitigation strategies, the P4-based solution excels in environments requiring rapid response times and minimal latency. This thesis contributes to the field of network security by providing a comprehensive framework for the deployment and evaluation of advanced DDoS detection systems. The integration of SDN, programmable data planes, and machine learning presents a promising approach to developing robust, scalable, and responsive network security solutions. The use of programmable networks like P4 in conjunction with machine learning models has shown to significantly enhance the capabilities of traditional network security systems, enabling them to adapt quickly to new and evolving threats.

## References

- [1] Cicflowmeter | canadian institute for cybersecurity, 2023.
- [2] 2024 DDoS Attack Trends | F5 Labs, 2024.
- [3] A Framework for In-network Inference using P4 | Proceedings of the 19th International Conference on Availability, Reliability and Security, 2024.
- [4] BACKORDERS | Proceedings of the 5th International Workshop on P4 in Europe, 2024.
- [5] hping | Linux, 2024.
- [6] Overview of DDoS Attack Research Under SDN. 2024.
- [7] Timur R. Bikbulatov and Ilya I. Kurochkin. Simulation of DDoS attack on software defined networks. *AIP Conf. Proc.*, 2181(1):020022, 2019.
- [8] Gibb Glen Izzard Martin McKeown Nick Rexford Jennifer Schlesinger Cole Talay coDan Vahdat Amin Varghese George Walker David Bosshart Pat, Daly Dan. P4. *ACM SIGCOMM Computer Communication Review*, 2014.
- [9] Christian Banse, Julian Schuette. A taxonomy-based approach for security in software-defined networking, 2017.
- [10] Khaled Zeraoulia Ferhat Mecerhed, Amel Benabdallah and Chafika Benzaïd. 3D-PM: A ML-powered Probabilistic Detection of DDoS Attacks in P4 Switches, 2023.
- [11] Abdelrahman Hegazy and Minar El-Aasser. Network Security Challenges and Counter-measures in SDN Environments. In *2021 Eighth International Conference on Software Defined Systems (SDS)*. IEEE.
- [12] Rakesh Kumar Jha Upena D. Dalal Idris Z. Bholebawa. Performance Analysis of Proposed Network Architecture: OpenFlow vs. Traditional Network. 2016.
- [13] Saqib Hakak Ali A. Ghorbani Iman Sharafaldin, Arash Habibi Lashkari. Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy. In *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE.
- [14] Mininet Project. Mininet Overview - Mininet, 2022.
- [15] Maaz Bin Ahmad Toqeer Mahmood Muhammad Awais, Muhammad Asif and Sundus Munir. Comparative Analysis of Traditional and Software Defined Networks, 2021.

- [16] Francesco Musumeci, Ali Can Fidanci, Francesco Paolucci, Filippo Cugini, and Massimo Tornatore. Machine-Learning-Enabled DDoS Attacks Detection in P4 Programmable Networks. *J. Netw. Syst. Manage.*, 30, 2022.
- [17] Open Networking Foundation. Sdn architecture, 2021.
- [18] Jacir L. Bordim Ranyelson N. Carvalho, Lucas R. Costa and Eduardo A. P. Alchieri. Detecting DDoS Attacks on SDN Data Plane with Machine Learning, 2021.
- [19] Ahmad Sanmorino and Setiadi Yazid. DDOS ATTACK DETECTION SIMULATION AND HANDLING MECHANISM. *Jurnal Ilmu Komputer dan Informasi*, 6(2):59–64, 2013.
- [20] Zhen Yao and Zheng Yan. Security in Software-Defined-Networking: A Survey-Web of Science Core Collection, 2016.
- [21] Shui Yu. An Overview of DDoS Attacks. In *Distributed Denial of Service Attack and Defense*. Springer, 2014.
- [22] Gu Lin Pan Shengli Guo Song Zeng, Deze. Scopus - document details - software defined networking i: Sdn. 2020.