AUTOMATION OF NETWORK FLOW ROUTING FOR CYBER ATTACK MITIGATION IN SDN

TANG JIA YANG

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITI MALAYA
KUALA LUMPUR

2025

AUTOMATION OF NETWORK FLOW ROUTING FOR CYBER ATTACK MITIGATION IN SDN

TANG JIA YANG

THESIS SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF COMPUTER SCIENCE (COMPUTER SYSTEM AND NETWORK)

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITI MALAYA
KUALA LUMPUR

2025

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: TANG JIA YANG

(I.C/Passport No:020316081109)

Matric No: U2102737

Name of Degree: Bachelor of Computer Science(Computer System and Network)

Title of Project Paper ("AUTOMATION OF NETWORK FLOW ROUTING FOR CYBER ATTACK MITIGATION IN SDN"):

Field of Study:

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date: 14-01-2025

ABSTRACT

Traditional networks rely on physical hardware components like switches and routers, which makes them static, decentralized, and difficult to scale or manage. These constraints frequently cause low traffic throughput and scalability concerns. SDN addresses these difficulties by separating the control and data planes, allowing for centralized, software-driven network management. However, the SDN controller's centralized management makes it a high-value target for attackers. To help avoid these risks, this paper introduces NetSecFlow, an automated solution for monitoring network security and actions triggering in SDN system. NetSecFlow collects real-time network telemetry and transmits it to an Elasticsearch-powered Security Information and Event Management (SIEM) system for threat detection. When detected, it communicates with the SDN controller (ONOS) via REST APIs to implement security measures such as path redirection, traffic blocking, and rate limitation. To improve detection accuracy and reduce false positives, the system combines Endpoint Detection and Response (EDR) and Intrusion Detection System (IDS) capabilities. NetSecFlow improves security by automatically recognising compromised assets and establishing segregated traffic pathways to reduce risks while maintaining normal operations. The system's robust logging mechanism captures all actions, allowing administrators to analyse and overrule automated answers based on business requirements. Testing in a simulated lab environment shows that NetSecFlow can detect and mitigate attacks in real time, while also increasing network performance and security. By combining automation, scalability, and robust security measures, NetSecFlow provides an innovative solution to reduce risks associated with SDN's centralized architecture, ensuring efficient and secure network administration.

Keywords: Software Defined Networking, Intent Based Routing, Security Automation

ACKNOWLEDGEMENTS

I would like to extend my heartfelt thanks to all who have played a pivotal role in my final year project research journey. My deepest gratitude is reserved for my supervisor, Associate Professor Dr Ling Teck Chaw, who gave me invaluable guidance and support along my journey. His expert insights have been particularly instrumental, especially in shaping my directions in the entire project. A lot of feedback and advice has been given by him that is useful to improve the project quality.

Next, I am profoundly thankful to OF@TEIN+++ for providing me all the hardware and software resources for my system development. I had learnt a lot while setting up the servers for my testing purposes.

I am also thankful to Faculty of Computer Science and Information Technology at Universiti Malaya. The faculty has provided me an outstanding academic environment with necessary resources and facilities that lead to my excel in this project. I really appreciate faculty's dedication to create a collaborative atmosphere, which has greatly enrich my research experience.

Last but not least, I wish to acknowledge the contributions of everyone who has supported me along this path. Your kindness and generosity have truly made a significant impact on me whether through advices, assistances or encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

API	:	Application Programming Interface
ACL	:	Access Control List
DDOS	:	Distributed Denial Of Service
ELK	:	Elasticsearch, Logstash, Kibana
EDR	:	Endpoint Detection and Response
IaC	:	Infrastructure as Code
IDS	:	Intrusion Detection System
K8s	:	Kubernetes
ML	:	Machine Learning
MITM	:	Man-In-The-Middle
NETCONF	:	Network Configuration Protocol
NFV	:	Network Function Virtualization
ONOS	:	Open Networking Operating System
OVS	:	OpenVSwitch
OVSDB	:	OpenVSwitch Database
P4	:	Programming Protocol-independent Packet Processors
PBR	:	Policy Based Routing
REST	:	Representational State Transfer
SDN	:	Software Defined Network
SIEM	:	Security Information and Events Management
TLS	:	Transport Layer Security
VM	:	Virtual Machine
YAML	:	YAML Ain’t Marking Language
ZTP	:	Zero Touch Provisioning

CHAPTER 1: INTRODUCTION AND BACKGROUND

1.1 Introduction

In traditional networks, the deployment of new infrastructure or any changes like network expansion, network decommissioning, and system upgrade must be done manually by humans. Due to the increasing number of internet users, a lot of new network devices need to be installed to meet the workload and demand of the users. A lot of human power is required for the device manual installation, and this is prone to errors. This diverts their attention from more important tasks like traffic optimization and network security monitoring. Any mistakes made could lead to serious consequences, especially in network security such as excessive permissions granted for the users. Since the scalability and availability of existing network architectures couldn't meet the demand of the users, a dynamic solution that can seamlessly introduce new devices is required – Software-Defined Network (SDN).

Software-Defined Networking (SDN) is a new paradigm in this field that allows control plane and data plane of the networking devices to be decoupled, allowing software application to programmatically manage the network. A network device is built up of two networking planes for forwarding which are data plane and the control plane. The control plane responsible of regulating how the packets are routed, the data plane is in charge of transiting data from one device to another. SDN operates by decoupling the network planes and leaves the network devices with only the data plane. The network devices must then establish a connection with a controller, which serves as their brain and gives them instructions on how to forward packets. With this knowledge in hand, the controller may then choose the most efficient path or most secure path to send a packet from its source to its destination while preserving the network's overall traffic throughput.

1.2 Research Motivation

Security and performance management in older network environments sometimes need manual intervention and static setups, which are incapable of scaling or adapting rapidly enough to meet the demands of modern, high-throughput networks. In contrast, SDNs allow you to programmatically regulate network traffic and dynamically configure devices. This feature provides a unique chance to create security measures that are not just reactive, but also proactive. The proposed system uses the inherent flexibility of SDNs to automatically detect anomalies in network traffic—such as unusual traffic flows, signs of unauthorized access, or potential data exfiltration attempts—and responds in real time by rerouting this suspicious traffic to specialized monitoring tools for further analysis. This not only helps in immediate threat mitigation but also aids in the detailed investigation of potential security breaches without disrupting normal network operations.

Furthermore, the integration of SIEM, EDR, and SDN controllers is projected to result in a highly customisable strategy that takes advantage of existing infrastructures and security expenditures. This connection is crucial because it guarantees that the orchestration system can easily communicate with and increase the capabilities of these tools, hence expanding their usefulness and boosting the network's overall security. This technology uses heuristic approaches to discover abnormalities in network behaviour and historical data, offering a reliable mechanism for spotting possible threats that standard signature-based systems may overlook.

The necessity for legacy systems and new technologies to coexist harmoniously is emphasized by the assumption of precise tool versions and compatibility. By meeting this requirement, the suggested solution not only improves security and performance, but also facilitates the transfer of organizations from old networking paradigms to SDNs. This is

especially critical in areas that need high levels of network stability and security, such as banking, healthcare, and government.

In summary, the motivation for building this security orchestration system arises from the necessity to handle the dual difficulties of ensuring robust security and high performance in SDNs. The system's capacity to detect and respond to abnormalities in real time, while seamlessly integrating with current tools and infrastructures, is a possible answer to these difficulties. By doing so, it contributes to the larger objective of making SDNs not only more controllable, but also more secure and efficient, therefore realizing their full potential to alter network administration and operations.

1.3 Problem Statement

The traditional network is now dealing with major concerns that threaten its effectiveness and security. First, there is a lack of dynamic security measures. The current network security architecture is unable to respond to real-time changes in network behaviour and developing threat patterns. This static approach dramatically increases the network's susceptibility during network attack since it is unable to adapt quickly or effectively to the intricacies of various threats as they emerge.

Furthermore, the security measures in place are inflexible and do not allow for scalability or agility. This inflexibility is especially troublesome during times of changing network traffic or when the network is under an attack. During such crucial periods, the inability to adapt security measures to changing conditions means that the network's defenses are frequently underutilized or overloaded, resulting in inefficient responses that do not match the intensity or specific type of the threat.

The traditional network's lengthy management procedures exacerbate these challenges. The use of manual procedures, as well as the complexity of administrative activities inside the network's architecture, contribute to poor reaction times, especially during cyber-attacks. The sluggishness in controlling and mitigating risks as they arise not only drains resources, but also hinders the network from operating at peak performance. The manual nature of these processes frequently causes a bottleneck effect, in which responses to security risks are delayed, allowing possible breaches to spread further than they would in a more agile setting.

These issues together degrade the network's capacity to safeguard data and sustain service continuity, especially in the face of advanced cyber-attacks. The network's current situation highlights the critical need for a revolutionary approach to network security and

administration—one that embraces automation, includes adaptive security rules, and simplifies management operations to improve responsiveness and efficiency. Implementing such modifications is critical not just for strengthening the network's defenses, but also for ensuring that it can support the company's activities safely and effectively in a rapidly changing digital environment.

1.4 Research Questions

To overcome the identified problems, several research questions must be raised and addressed:

- How can network programmability in SDN improve security posture for a company?
- How can security policy be designed to handle varying levels of network traffic and threat intensities during cyber-attacks?
- How can network management tasks be simplified to improve responsiveness and efficiency during cyber-attacks?

1.5 Research Aim and Objectives

The purpose of this research is to develop a system that can automatically detect anomalies in a network and respond to it by modifying certain traffic paths for further analysis to reduce network administrator burden. To develop a system that fulfills the previously mentioned requirements, the following objectives need to be achieved:

- To improve the organization's security posture by deploying dynamic security measures that automatically adapt to real-time network and threat behaviours
- To create scalable, adaptive security rules inside the SDN architecture that react to different network traffic and threat levels during attack and peak periods.
- To reduce the need for manual intervention and speed up response times during cyberattacks and streamline network management tasks

1.6 Project Schedule

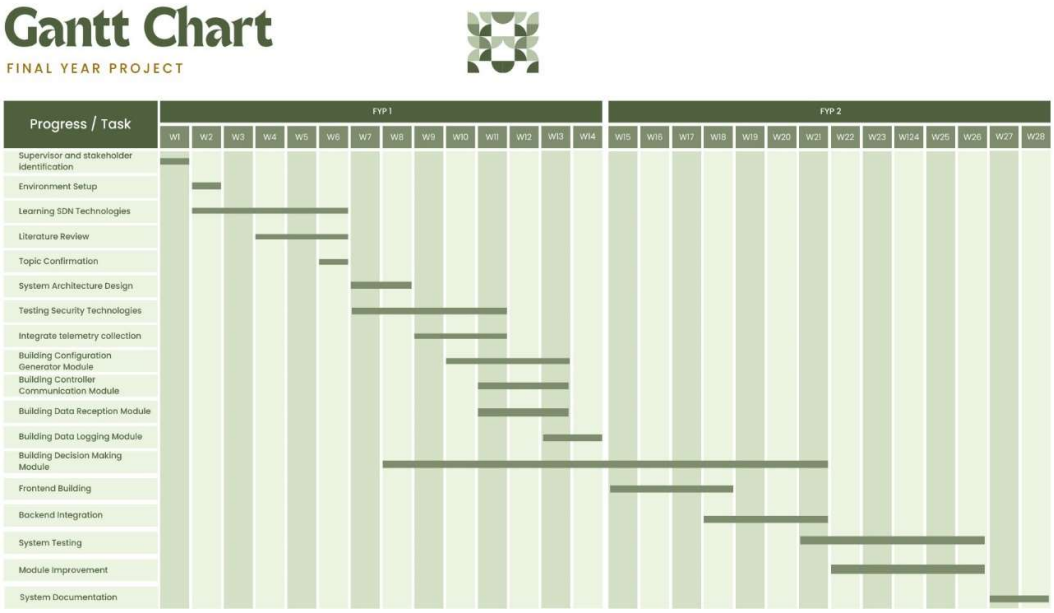


Figure 1 Timeline for Project

Above table is the gantt chart for the project throughout two semesters comprising of 28 weeks. This timeline are particularly important to follow in order to finish the works within the deadline.

1.7 Scope of Work

The security orchestration system is a proof-of-concept work, so certain assumptions need to be made:

- All users of the system will have predefined versions and types of SIEM, EDR, and SDN controller tools installed and operational in their network environment.
- The user must have basic understanding in managing and operating advanced network security tools.
- The system by default doesn’t cover advanced persistent threat in a network.

1.8 Thesis Outline

The thesis covers five chapters. Chapter 1 is on introduction and background of the project which includes research motivation, problem statement, objectives, research aim, project schedule and scope of work. Chapter 2 is about literature review of similar research on security orchestration and path modification. Chapter 3 displays the proposed system, requirements, design and methodology used. Chapter 4 describes implementation and testing done on the system. Chapter 5 delivers the conclusion obtained from the study and gives an insight into potential future works.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

In this chapter, we will review the existing literature on the main security issues and solutions within the SDN architecture. We will explore various studies and systems that have been proposed or implemented to address these challenges. The review will be structured to highlight key areas of research, compare existing tools, and identify gaps in the current body of knowledge.

In section 2.2, research papers and documentation related to SDN security, existing products, gaps identified will be reviewed for project planning. In this section, we delve into the conceptualization of various products and systems designed to enhance security in SDN architectures. We will analyze different research studies and their proposed solutions to understand the state-of-the-art in SDN security.

In section 2.3, we compare different tools and technologies used in SDN security. We will justify the selection of specific tools based on their capabilities, performance, and relevance to the security issues identified in the previous sections. This analysis will provide a rationale for the tools chosen for the development of NetSecFlow.

In section 2.4, we will wrap up this chapter and summarize our findings for the project.

2.2 Product Ideation

This section will be discussing the research papers, articles and documentations that will provide the ideas on improvement needed on this project. This paper will further talk on the gap on the current research and how this project going to improve on that.

2.2.1 Title: Main Security Issues and Solutions for The SDN Architecture

Jiménez, M. B., Fernández, D., Rivadeneira, J. E., Bellido, L. L., & Cárdenas, A. F. P. (2021). A survey of the main security issues and solutions for the SDN architecture. IEEE Access, 9, 122016–122038. https://doi.org/10.1109/access.2021.3109564

The first article conducts a thorough study of the security vulnerabilities found in the various levels and interfaces of the Software Defined Network (SDN) architecture. The authors summarize the numerous security issues that exist across the northbound, southbound, and east-west interfaces of SDNs, and give a variety of solutions presented in current literature. The study adopts the STRIDE technique, which is a systematic way to identify and address possible security risks such as spoofing, tampering, repudiation, information leakage, denial of service, and elevation of privilege. This scientific examination emphasizes the paper's key contribution of methodically organizing the many security concerns and their related mitigation solutions, making it a valuable resource for understanding and safeguarding SDN settings.

2.2.2 Title: Automatic Mapping of Cyber Security Requirements to Support Network Slicing in Software-Defined Networks

M. Ehrlich, L. Wisniewski, H. Trsek, D. Mahrenholz and J. Jasperneite, "Automatic mapping of cyber security requirements to support network slicing in software-defined networks," 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Limassol, Cyprus, 2017, pp. 1-4, doi: 10.1109/ETFA.2017.8247728. keywords: {Computer security;Quality of service;Standards;Automation;Industries},

The second study investigates a unique method for incorporating cybersecurity Quality of Service (QoS) criteria into network management systems, with an emphasis on SDN

applications in network slicing and industrial automation. It expands on the ISO/IEC 62443 standard, intending to achieve automated, continuous, and machine-readable security enforcement in network management. The primary contribution of this study is the use of standardized cybersecurity QoS criteria, which allow for continuous modification of security policies in response to emerging threats and changing network conditions. This study emphasizes the need of a technical framework that can be tailored to both old and developing technologies, hence improving overall network security.

2.2.3 Title: Enhancing Network Intrusion Recovery in SDN with Machine Learning

Hammad, Mohamed & Hewahi, Nabil & Elmedany, Wael. (2023). Enhancing Network Intrusion Recovery in SDN with machine learning: an innovative approach. Arab Journal of Basic and Applied Sciences. 30. 561-572. 10.1080/25765299.2023.2261219.

The third study describes a Machine Learning-based Network Intrusion Recovery (MLBNIR) technique that aims to improve intrusion recovery procedures inside SDN frameworks. The suggested technique uses machine learning algorithms to make intelligent real-time judgements regarding network setups and path selections, optimizing network intrusion responses. The key advantage of this technique is its ability to optimize network resource utilization while taking into account current network conditions and traffic patterns. This study makes major contributions to the area by demonstrating how machine learning may be used to increase the resilience and efficiency of SDNs in the face of security threats.

2.2.4 Title: Cisco and Juniper Current Solutions

Software-defined networking (SDN) definition (2023) Cisco. Available at: https://www.cisco.com/c/en/us/solutions/software-defined-networking/overview.html#~stickynav=3 (Accessed: 11 May 2024).

SDN and orchestration: Juniper Networks Us (no date) Juniper Networks. Available at: https://www.juniper.net/us/en/products/sdn-and-orchestration.html (Accessed: 11 May 2024).

For Cisco, the documentation discusses how Cisco Talos was integrated with Cisco's Application Centric Infrastructure (ACI) and Digital Network Architecture (DNA) Centers to improve real-time threat intelligence. This connection improves automated security responses and boosts data center visibility, enabling proactive enforcement and anomaly detection in encrypted traffic across organizational networks. The study emphasizes how such integration is dependent on specialized Cisco hardware and knowledge, particularly the Encrypted Traffic Analytics function for malware detection in encrypted traffic, which is a notable feature of this technology.

For Juniper, the documentation discusses the company's creation of a flexible, policy-driven security framework through Contrail Networking, which is complemented with machine learning-based threat detection via Juniper Advanced Threat Prevention (ATP). This system enables secure network segmentation and uses behavioural analytics to proactively control attacks, but it requires Juniper hardware and Junos OS experience. The paper demonstrates how Juniper's methodology secures network infrastructures while also ensuring that security policies are smoothly incorporated into network management procedures.

2.2.5 System Comparison

Metric	Classic Solution(Cicso + Juniper Solution)	My Solution
Device Supported	Proprietary to vendor	Open-Source Support custom-build solution
Cost-Effectiveness	Initial Cost higher due to licensing fee	Potentially lower initial costs due to the use of open-source software
Resource Efficiency and Reliability	Optimized for performance out of the box but may not be as adaptable to unique network configurations	Potentially higher resource efficiency as it can be finely tuned to specific network needs
Integration Capabilities	Limited to specific ecosystem	Can easily integrate with various existing system and technologies

Table 1 System Comparison

2.2.6 Research Directions and Gaps

Current SDN focuses on increasing network flexibility, programmability, and centralized control while resolving traditional network restrictions. Despite these advances, substantial gaps exist in maintaining adequate security for SDN settings. Key research topics include improving the security of the SDN controller, which is a significant vulnerability point, and creating sophisticated intrusion detection and prevention systems designed particularly for SDN architectures. Furthermore, combining

Deep Packet Inspection (DPI) with machine learning approaches for real-time threat identification and response offers exciting possibilities. However, issues remain in balancing security measures with network performance and scalability. There is a need for complete solutions that can dynamically respond to changing threats while maintaining network efficiency.

2.3 Tools Comparison and Justification

This section will be discussing the tools that are used to build this project for testbed simulation. These tools are chosen based on its features and comparison with current available technologies. Justification made is based on the product documentation.

2.3.1 Operating System

Ubuntu documentation. Available at: https://docs.ubuntu.com/ (Accessed: 14 June 2024).

The NetSecFlow system was developed using Ubuntu 22.04.4 LTS because to its stability, security, and long-term support, assuring dependable and up-to-date performance. This version provides wide interoperability with important networking tools and software, as well as powerful security capabilities required for a network security project. Furthermore, Ubuntu's open-source nature, ease of use, and large support community make it an affordable and developer-friendly environment. These characteristics work together to form a solid basis for creating and implementing the NetSecFlow system in an efficient and secure manner.

2.3.2 Programming Language

Python 3.12.4 documentation (no date) 3.12.4 Documentation. Available at: https://docs.python.org/3/ (Accessed: 14 June 2024).

For the development of the NetSecFlow system, Python 3.10.12 has been chosen for implementing all the basic backend logic. This version of Python ensures compatibility with modern libraries and tools, offering advanced features and improvements over previous versions. Its robust performance and extensive support for various protocols provide greater flexibility in network administration, making it an ideal choice for developing a network security assessment system. Additionally, Python 3.10.12’s stability and widespread community support ensure that any issues can be quickly resolved, facilitating smooth and efficient development. Comparing to other

programming languages like Java, Python offer extensive libraries which simplify the coding process of the backend logic.

2.3.3 Containerization Tool

Home (2024) Docker Documentation. Available at: https://docs.docker.com/ (Accessed: 14 June 2024).

Docker was chosen as the containerization solution for the NetSecFlow system due to its extensive usage, ease of use, and solid ecosystem. Docker provides a uniform and efficient environment for deploying and maintaining applications, ensuring that the NetSecFlow system operates consistently in a variety of contexts. Compared to alternative containerization systems such as Kubernetes and Podman, Docker shines in terms of simplicity and usability, making it more accessible to developers and easier to incorporate into current processes. While Kubernetes has extensive orchestration capabilities for large-scale deployments, Docker's integration with Docker Compose streamlines multi-container applications, making it appropriate for the NetSecFlow project's requirements. Podman, on the other hand, provides rootless containers and improved security by design; but Docker's stable ecosystem, copious documentation, and strong community support make it the best choice for this project, assuring a smooth and quick development experience.

2.3.4 SDN Controller

Foundation, O.N. (no date) Open network operating system (ONOS), onosproject. Available at: https://docs.onosproject.org/ (Accessed: 14 June 2024).

ONOS 2.5.0 was chosen as the operating system for the NetSecFlow system because it outperformed its competition, OpenDaylight, in terms of latency and throughput. ONOS' architecture is designed to manage huge network topologies effectively, making it an excellent solution for complicated network settings. The improved speed of ONOS 2.5.0 assures faster reaction times and more data processing capabilities, which are critical

for a network security assessment system like NetSecFlow. Furthermore, ONOS's excellent support for scalability and active community gives additional benefits, ensuring robust and dependable network administration.

2.3.5 SIEM tool

Documentation (no date) Elasticsearch Platform - Find real-time answers at scale. Available at: https://www.elastic.co/docs (Accessed: 14 June 2024).

Documentation (no date) Documentation - Splunk Documentation. Available at: https://docs.splunk.com/Documentation (Accessed: 14 June 2024).

The ELK Stack (Elasticsearch, Logstash, and Kibana) was chosen for its robust real-time data processing, indexing, and visualization capabilities. It is an open-source solution, which reduces its costs in comparison to its competitor, Splunk. While Splunk has a more comprehensive set of functionalities out of the box and a more intuitive user interface, ELK offers better flexibility and scalability for customized solutions, particularly for low-budget projects. ELK also requires less processing power compared to Splunk. The ELK version I am choosing is 7.17.13.

2.3.6 NIDS

Suricata User guide (no date) Suricata User Guide - Suricata 8.0.0-dev documentation. Available at: https://docs.suricata.io/en/latest/ (Accessed: 14 June 2024).

Snort setup guides for emerging threats prevention. Available at: https://www.snort.org/documents (Accessed: 14 June 2024).

Suricata is recommended because of its multi-threaded architecture, which allows it to manage larger traffic loads and performs better on multi-core platforms. It supports a greater variety of protocols and includes sophisticated capabilities such as file extraction and MD5 checksum. Snort, a long-standing and extensively used IDS/IPS, runs in a single-threaded mode, which might be a restriction in high-performance systems. Suricata's ability to do both inline intrusion prevention and deep packet inspection makes

it a more versatile option. Suricata has more updated signature compared to snort(community edition) because it is a community based project. The suricata version I am using is 6.0.4.

2.3.7 Network Switch

Getting started¶ (no date) Getting Started - Open vSwitch 3.3.90 documentation. Available at: https://docs.openvswitch.org/en/latest/intro/ (Accessed: 14 June 2024).

The NetSecFlow solution was developed using Open vSwitch (OVS) due of its extensive networking features and fast speed. OVS provides comprehensive virtual networking capabilities, which is critical for NetSecFlow's dynamic and scalable network environments. Its interoperability with multiple virtualization platforms and interaction with common cloud management systems make it an adaptable option. Furthermore, OVS' support for fine-grained traffic control and monitoring enables exact network traffic management and analysis, which is well aligned with NetSecFlow's aims. The active development community and extensive documentation help to guarantee that OVS remains a dependable and safe option for network virtualization and control.

2.3.8 EDR

Wazuh (no date) WAZUH documentation, Wazuh documentation. Available at: https://documentation.wazuh.com/current/index.html (Accessed: 14 June 2024).

Wazuh was chosen due to its extensive open-source security monitoring features, which include intrusion detection, log analysis, and incident response. Compared to existing Endpoint Detection and Response (EDR) systems, Wazuh provides a highly customizable and scalable platform that seamlessly connects with the ELK stack, resulting in a unified security monitoring solution. The wazuh version I am using is 4.5.4

2.4 Summary

The NetSecFlow technology was developed after extensive research to overcome current network security weaknesses. The selection of these technologies is guided by research into SDN security concerns, automated cybersecurity needs, and machine learning-based intrusion recovery, guaranteeing that NetSecFlow effectively solves current vulnerabilities and enhances network security. After reviewing the current product gaps and tools selection, in next chapter, we will look at how to integrate them in a system design that will fulfill the gaps in current system.

CHAPTER 3: THE PROPOSED SOLUTION AND ITS DESIGN

3.1 Introduction

In the previous chapter, a literature review was conducted on research that is based on security to discover any existing gaps. Although there are similar technologies proposed by the vendor, the system that I proposed is vendor-neutral approach that integrates seamlessly with a broader range of network devices and management tools, enhancing interoperability across diverse network environments. Therefore, in this chapter, we will propose and design NetSecFlow, a system that will integrate current available technologies to address the proprietary issue in SDN environment.

In Section 3.2, the research methodology used to conduct this project will be discussed. In Section 3.3, software development methodology chosen and phases for the methodology will be reviewed. Section 3.4 explains the hardware and software requirements followed by the functional and non-functional requirements. Finally, the system architecture and deployment strategy will be introduced in Section 3.5.

3.2 Research Methodology

This NetSecFlow project's research methodology includes an organized approach to testing and validating the effectiveness of cyber threat mitigation using Software-Defined Networking (SDN). The project is built around a testbed of two servers, each equipped with critical tools and technologies for simulating and analyzing network environments under various cyber-attack scenarios.

For environment setup, two high-performance servers are set up with Ubuntu 22.04.4 LTS to offer a reliable and secure operating system. These servers run the ELK Stack (Elasticsearch, Logstash, and Kibana), the ONOS SDN controller, Wazuh for security monitoring, and Docker for containerization. Python 3.10.12 is used for backend logic

implementation, using its rich library support to provide network security solutions. ReactJS is used for my frontend development

After finishing setting up the environment, software development phases will start taking into account. Using the data acquired by the security tools, a custom application is created to automate flow management. This program uses information from the security tools to dynamically modify network flows, improving the system's response to recognized threats. The application communicates with the ONOS controller in real time to apply these modifications, assuring the best network performance and security.

The next stage would be simulation and testing phase. Various cyber-attack scenarios, such as Distributed Denial of Service (DDoS), spoofing, and data breaches, are simulated to assess the system's reaction and mitigation capabilities. The bespoke program automatically adjusts network flows based on real-time data from security tools, showing its efficiency in mitigating various threats. The performance of ONOS's SDN environment is closely monitored, with data gathering and analysis enabled by Wazuh , Suricata, ELK Stack and other tools.

After the testing phase, the next stage would be evaluation stage. The process includes constant review and iteration. Following each round of testing, the data are analyzed to determine the efficacy of the mitigation techniques and the performance of the customized application. Based on the findings, adjustments are made to improve the system's performance and security. This iterative approach guarantees that the final implementation of NetSecFlow is strong and capable of mitigating a wide range of cyber-attack.

This organized method, which includes a controlled testbed environment, systematic analysis, and a bespoke application for automated flow management, enables a full

evaluation of cyber-attack mitigation measures based on SDN, offering new insights to the field of network security.

3.3 Software Development

For software development, the methodology that is chosen is Agile. The development that follows Agile techniques encourages flexibility and iterative growth via regular feedback loops. This technique allows for quick prototyping and continual system enhancement, guaranteeing that the solution is successful and efficient under a variety of simulated network settings and cyberattack scenarios. The objective is to build a strong system capable of making dynamic routing adjustments and adjusting security measures in real time to guard against attackers.

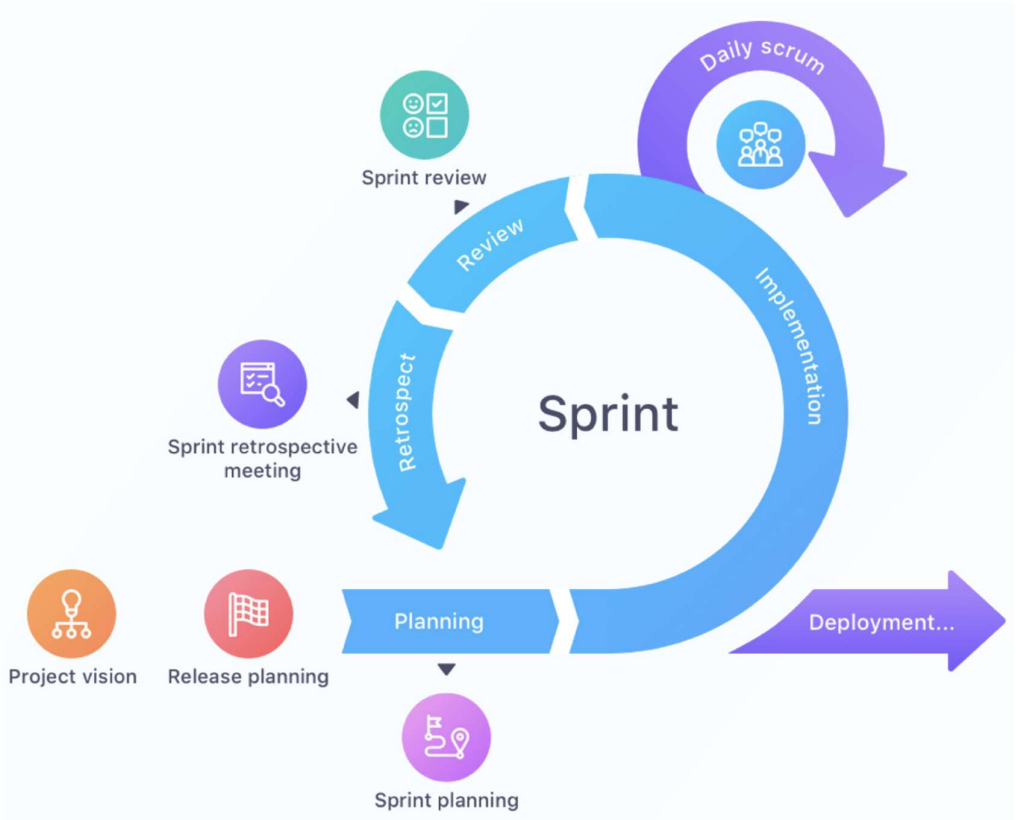


Figure 2 Agile Phases Diagram

Agile contains multiple critical phases, including Sprint Planning, Iterative Development, Integration and Testing, Review and Adaptation, and Release and Refinement. Sprint Planning is the first step that focuses on identifying precise targets for improving dynamic threat response capabilities and allocating development tasks related to security automation. Clear objectives are established to direct development activities during the sprint. The next step is iterative development, in which the project requires fast prototypes of traffic analysis and routing modifications. Regular meetings with supervisors and collaborators give critical real-time input that influences the development process and ensures that prototypes satisfy functional requirements efficiently.

In the Integration&Testing phase, the produced features are regularly integrated into the SDN environment, followed by thorough testing to assess performance. These tests provide feedback that is utilized to modify and optimize the system, increasing its dependability and efficacy. Each sprint in the Review&Adaptation phase finishes with a review session during which the development team collects and analyses input. This method assists in identifying any necessary modifications, which are subsequently applied in succeeding sprints to improve the system's functionality and performance.

The final phase is Release&Refinement comprises the gradual deployment of system upgrades, allowing for continual refining in response to user feedback and emerging threats. This continuous optimization procedure ensures that the system stays successful in changing network settings and against sophisticated cyber-attacks.

3.4 Requirements

The next section will examine NetSecFlow's hardware and software requirements. The essential software will be briefly introduced and shown how to utilize it in NetSecFlow. Then, the functional and non-functional criteria that must be met before the system can be considered complete will be suggested.

3.4.1 Hardware

NetSecFlow requires the use of SDN controllers, OpenVSwitch and other security software. Hence, machines with appropriate specifications are required to support the workload.

Hardware	Specification
Processor	Intel Xeon Processor E5645, 2.4GHz x6
Memory	16 Gigabytes or higher
Hard Disk	50GB of available hard disk space
Video Card	Intel UHD Graphics 620
Display Resolution	1024 x 768 or higher

Table 2 Hardware Requirements

3.4.2 Software Requirements

Tool	Description	Version
Ubuntu	Operating System	22.04.4 LTS
Python	Programming Language	3.10.12
Docker	Containerization Tool	24.0.5
ONOS	SDN Controller	2.5.0
ELK stack	SIEM tool	7.17.13
Suricata	NIDS	6.0.4
OpenVSwitch	Network Switch	2.17.9
Wazuh	EDR	4.5.4

Table 3 Software Requirements

3.4.3 Functional Requirements

Functional requirements provide the characteristics and functions that the system must provide to be effective. To begin, the system must detect traffic anomalies to identify abnormal patterns and potential network threats. In addition, it should work with intrusion detection systems to improve its capacity to detect and respond to security breaches. Another important feature is automatic configuration, which allows the system to automatically alter settings and policies without requiring operator interaction. Logging and monitoring are required for keeping a thorough record of network activity, which is necessary for troubleshooting and auditing. The system must also offer route and traffic modification, which allows for the redirection of data flows to improve network performance and security. Finally, it should enable intent-based routing, allowing users to define routing policies based on their specific requirements, thus providing a flexible and customizable network management solution.

3.4.4 Non-Functional Requirements

Non-functional requirements are features that characterize the system's performance and other non-behavioral characteristics. One of the key non-functional requirements is ease of deployment, which ensures that the system can be installed quickly and effectively with little disturbance to existing activities. High dependability is also required, particularly in the detection of threats, to enable consistent and accurate identification of security risks. Another essential requirement is compatibility with existing network infrastructure, which allows the system to connect easily with existing technologies and devices without requiring substantial changes. Finally, the system must have quick speed, allowing it to analyze data and respond to network events in real time, providing maximum network efficiency and security.

3.5 System Design

In the previous section, a proposal of the requirements for NetSecFlow was presented. Now, this section will deliver the system design which includes the architecture of the system.

3.5.1 System Architecture

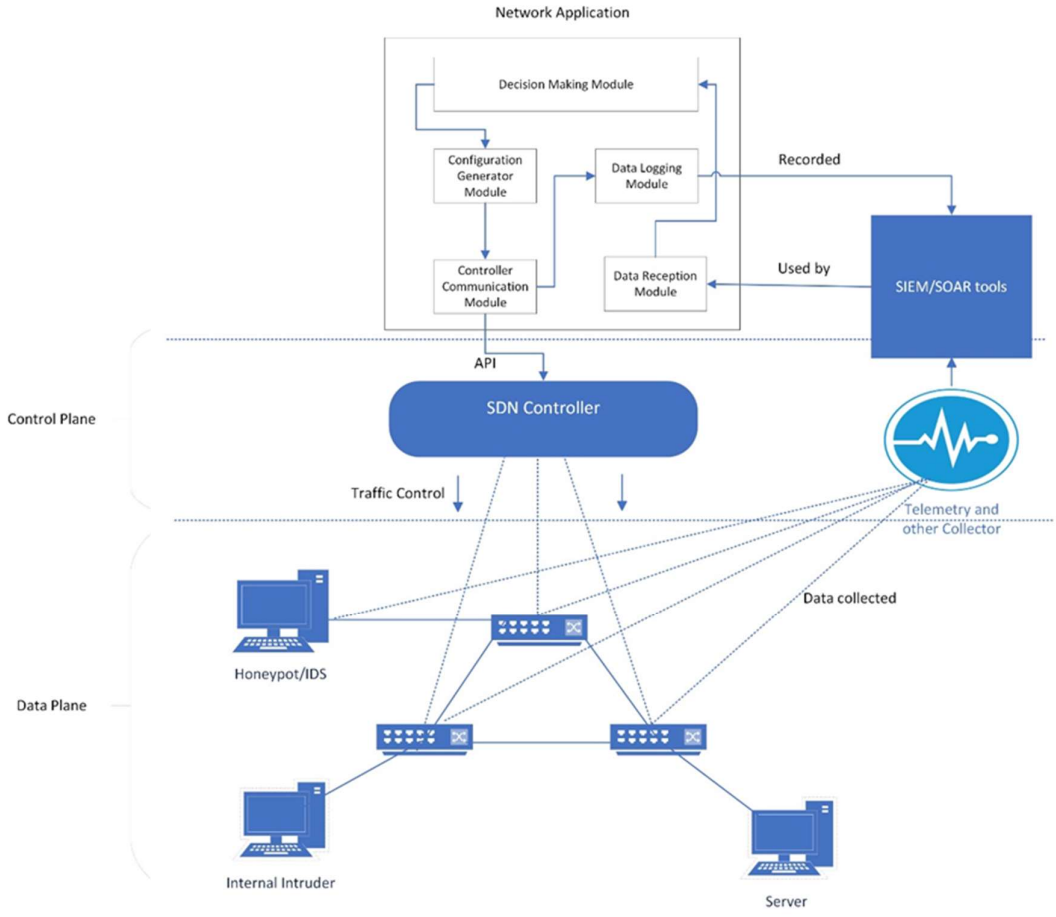


Figure 3 System Architecture and Design

The architectural diagram depicts a sophisticated system that detects security vulnerabilities and automatically adjusts traffic flow in a Software Defined Network (SDN). The SDN Controller is at the heart of the system, communicating with both the control and data planes. The control plane includes a Network Application, which consists of several critical modules: the Decision Making Module, Configuration Generator

Module, Controller Communication Module, and Data Logging Module. These modules collaborate to dynamically control network settings using security data and analytics.

On the data plane, the SDN Controller directs and controls network traffic, which may be rerouted through various network devices such as Honeypots/IDS in order to detect and mitigate any risks. It also manages traffic from internal network components such as servers and possibly compromised internal intruders. Furthermore, telemetry and other data collectors provide feedback to the SDN Controller, allowing it to make more intelligent traffic routing decisions in response to observed network behaviour or threats.

Overall, this design provides a responsive and adaptive network security management system inside an SDN environment, automating procedures to resolve security concerns quickly and without manual involvement, therefore improving both network security and operational efficiency.

3.5.2 Application Architecture

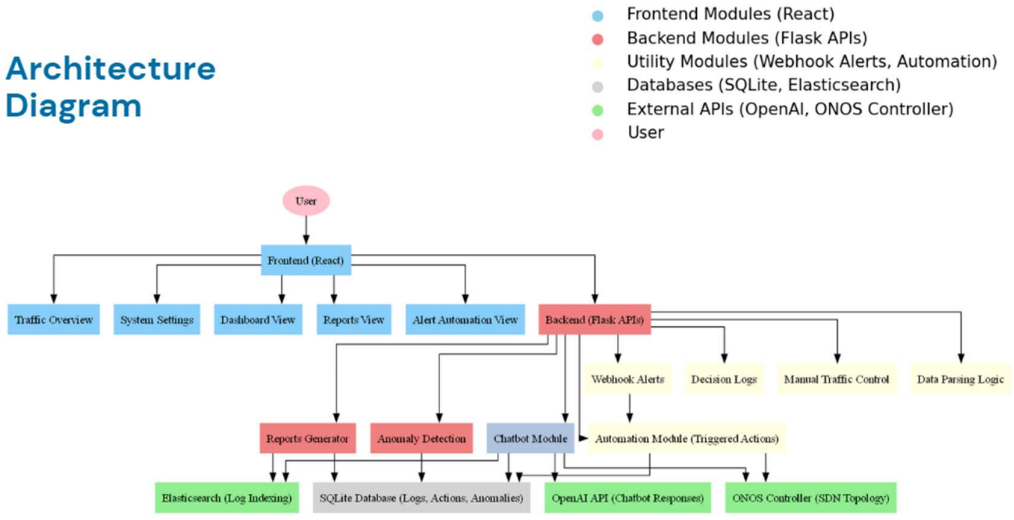


Figure 4 Application Architecture and Design

The design highlights the interplay between the frontend, built with React, and the backend, powered by Flask APIs, to ensure real-time processing, dynamic visualization, and robust system responsiveness. The frontend manages user interaction and visual representation, providing intuitive navigation through modules such as traffic overview, system settings, dashboard view, and alert automation.

At its core, the backend facilitates advanced functionality by integrating utility modules like webhook alerts and automation. These modules are essential for triggering alerts, anomaly detection, and managing automated responses to network events. A distinct feature of this architecture is its ability to incorporate external APIs, such as the OpenAI API for chatbot responses and the ONOS controller for Software Defined Network (SDN) topology management. This integration enables a blend of intelligent automation and network visualization, enhancing decision-making processes.

The database layer is another pivotal element in this architecture. SQLite is utilized for storing logs, actions, and anomalies, while Elasticsearch provides efficient log indexing and searching capabilities. These databases support data integrity, scalability, and fast retrieval, ensuring that the system can handle large datasets and maintain operational efficiency. Reports generation and anomaly detection modules rely on this robust data storage to provide insightful analytics and timely identification of irregularities.

Additionally, the application architecture promotes modularity and scalability by separating the frontend modules, backend APIs, and utility modules. This separation ensures ease of maintenance and future adaptability, making the system flexible for incorporating additional features. The inclusion of automation modules and decision-making logic further enhances the system’s ability to streamline manual tasks, enabling faster resolution of network issues and improved overall performance.

By combining user-centric design, backend intelligence, and advanced automation tools, this application architecture provides a comprehensive solution for managing complex systems. Its layered approach ensures that each module functions cohesively, supporting users with reliable data visualization, real-time monitoring, and actionable insights.

3.5.3 Decision Making Logic

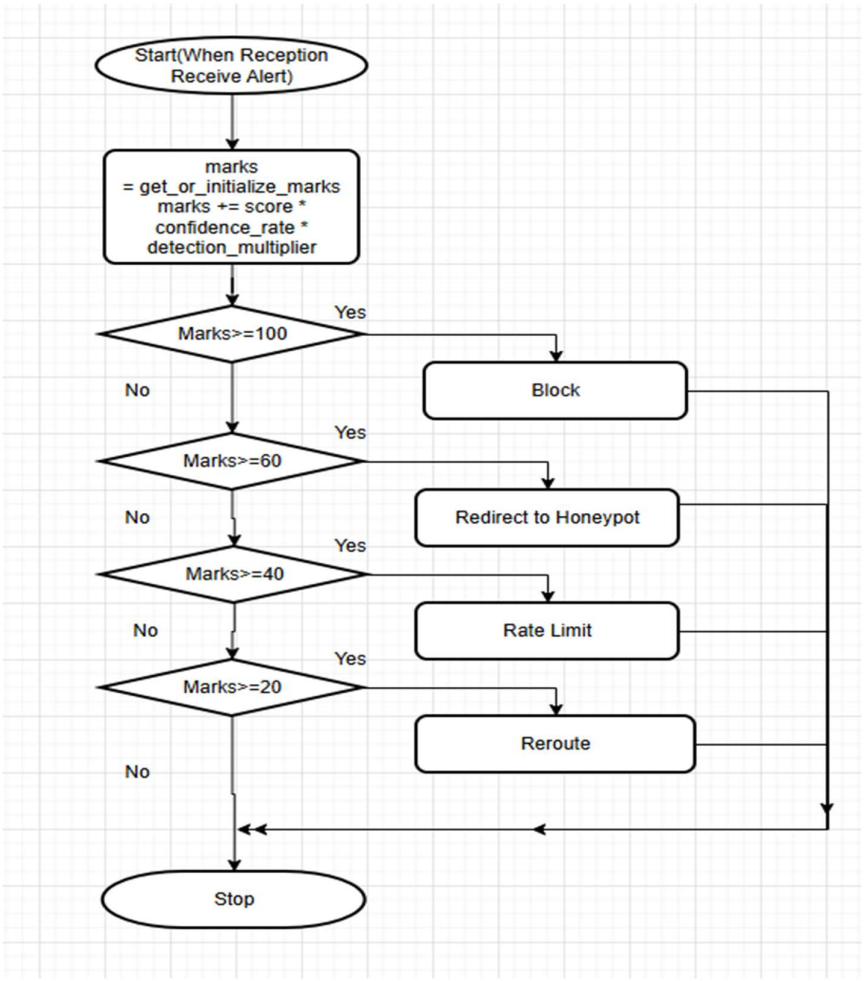


Figure 5 Decision Making Logic

The flowchart illustrates the decision-making process of a reception module for handling alerts based on their severity and context. This process evaluates the risk score

associated with an alert and assigns actions accordingly to mitigate potential threats effectively.

When an alert is received, the system initializes or retrieves the existing marks for the source. The marks are incremented based on the severity score, confidence rate, and detection multiplier associated with the alert. This dynamic scoring system ensures that alerts are evaluated comprehensively, reflecting their potential impact on the network.

When the marks assigned to a source reach or exceed 100, the system takes immediate action to block the source. This is the most severe measure implemented to neutralize high-risk threats and prevent further malicious activity. Blocking the source ensures that no additional damage can be caused, thereby safeguarding the network's integrity.

If the marks fall between 60 and 100, the system redirects the source to a honeypot. This strategy isolates the potential threat from the main network, preventing it from disrupting regular operations. At the same time, the honeypot allows the system to gather valuable intelligence about the behavior of the threat, which can inform future security measures and enhance overall threat detection capabilities.

For sources with marks between 40 and 60, the system applies rate limiting. This action restricts the traffic rate from the source, effectively minimizing its impact on the network while maintaining minimal connectivity. By throttling the traffic, the system ensures that the potential threat is contained without completely cutting off communication, allowing for a balanced response.

Lastly, when the marks are between 20 and 40, the system reroutes the traffic through an intermediate path. This approach reduces the risk posed by the suspicious traffic by directing it along a less critical path while utilizing reactive routing mechanisms. This action minimizes the potential harm caused by the traffic while maintaining network

functionality. Together, these tiered actions form a comprehensive strategy for managing network threats effectively.

If the marks do not meet any of the thresholds, no action is taken, and the system halts the process. This structured approach allows the system to adapt its response based on the severity of the detected alert, ensuring a balance between security and network performance. By implementing a tiered response mechanism, the system enhances its efficiency in threat management while minimizing disruptions to legitimate network activities.

3.5.4 Deployment Strategy

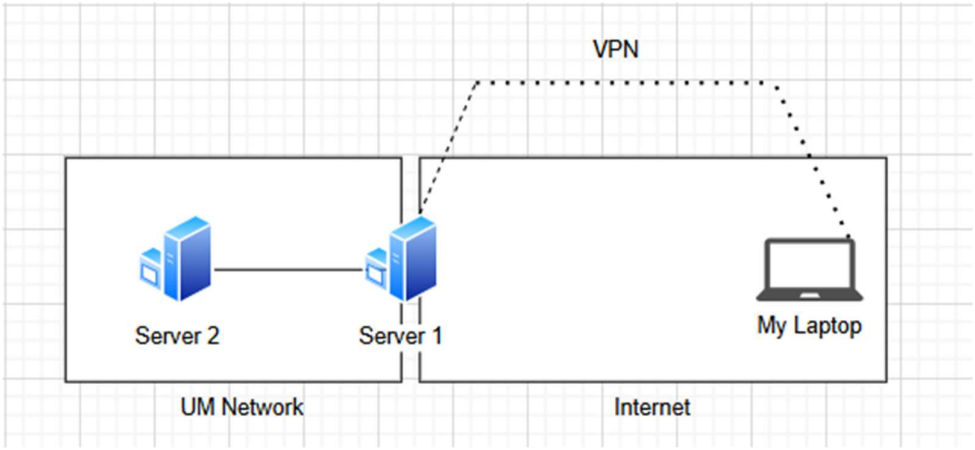


Figure 6 Test Environment Deployment

The deployment strategy for NetSecFlow leverages a combination of high-performance university servers and a personal laptop to create a robust, scalable, and secure environment for real-time network security management.

Server 1, dedicated to running the ELK stack, is optimized for resource-intensive operations such as real-time log aggregation, threat analysis, and data visualization. This server processes and correlates logs from multiple detection tools, including Suricata and

Wazuh, enabling comprehensive threat detection and network monitoring. Its high processing power and substantial storage capacity ensure smooth operation and reliable handling of large volumes of data.

Server 2 hosts the Mininet network emulation environment, complemented by various virtual machines (VMs) managed through KVM. These VMs emulate critical Software-Defined Networking (SDN) components, including the ONOS controller, Open vSwitch instances, and virtual devices, providing a controlled environment for testing network traffic and simulating cyberattacks. The server's versatility enables it to mirror real-world network scenarios, facilitating rigorous testing and validation of security measures.

The personal laptop serves as the control node, hosting the web application that acts as the system's decision-making and monitoring interface. This application interacts with the ELK server to retrieve real-time threat analysis data, processes the information through a decision-making module, and sends traffic management instructions to the SDN controller on Server 2. By centralizing control on the laptop, the system ensures efficient coordination across its components while maintaining a user-friendly interface for administrators.

This deployment strategy effectively balances computational efficiency, scalability, and operational precision, providing a comprehensive framework for securing SDN environments in real-time.

3.6 Summary

This chapter we are discussing about the project architecture diagram that try to fulfill the requirements in current available system. In next chapter, proposed NetSecFlow will be implemented and evaluated, the chapter will include the detailed process of the implementation and evaluation.

CHAPTER 4: NETSECFLOW: DEPLOYMENT IMPLEMENTATION AND EVALUATION

4.1 Introduction

In the previous chapter, the design and architecture of the proposed NetSecFlow system were detailed, alongside the methodologies and requirements that form the foundation of its development. The chapter also introduced the deployment strategy, ensuring that the system's components are aligned with the goals of vendor neutrality, interoperability, and scalability in Software-Defined Networking (SDN) environments.

This chapter will focus on the deployment, implementation, and evaluation of NetSecFlow. Section 4.2 will present the user interface of the system, highlighting its design and functionalities to enhance user experience. Section 4.3 will detail the deployment process, including the setup of hardware and software components as outlined in the previous chapter. Section 4.4 will delve into the implementation of NetSecFlow, describing the integration of technologies, configurations, and workflows required to operationalize the system. Section 4.5 will evaluate the system's performance and effectiveness, with key metrics and benchmarks to validate its functionality. Finally, Section 4.6 will provide a discussion of the results, including challenges encountered and potential areas for improvement.

The findings and analysis presented in this chapter will demonstrate how NetSecFlow achieves its objectives of enhancing network security and addressing proprietary limitations in SDN environments. This sets the stage for future considerations and developments discussed in subsequent chapters.

4.2 User Interface

In this section, the user interface of NetSecFlow, ELK and ONOS will be presented.

4.2.1 Dashboard Page

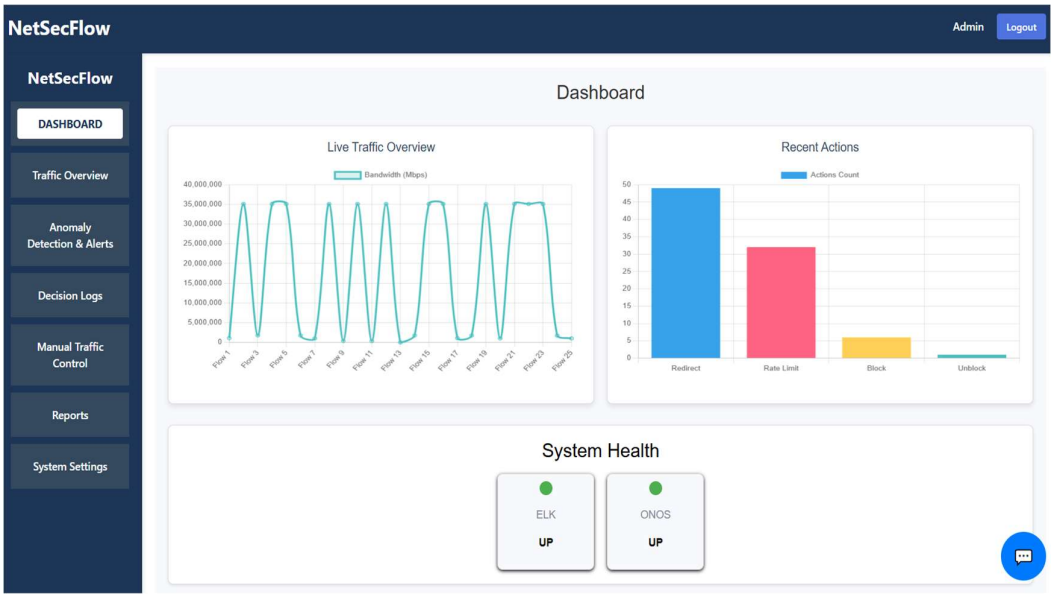


Figure 7 NetSecFlow Dashboard Page

This page provides an overview of the system, displaying key statistics and metrics related to traffic, application status and security events in real time.

4.2.2 Traffic Overview Page

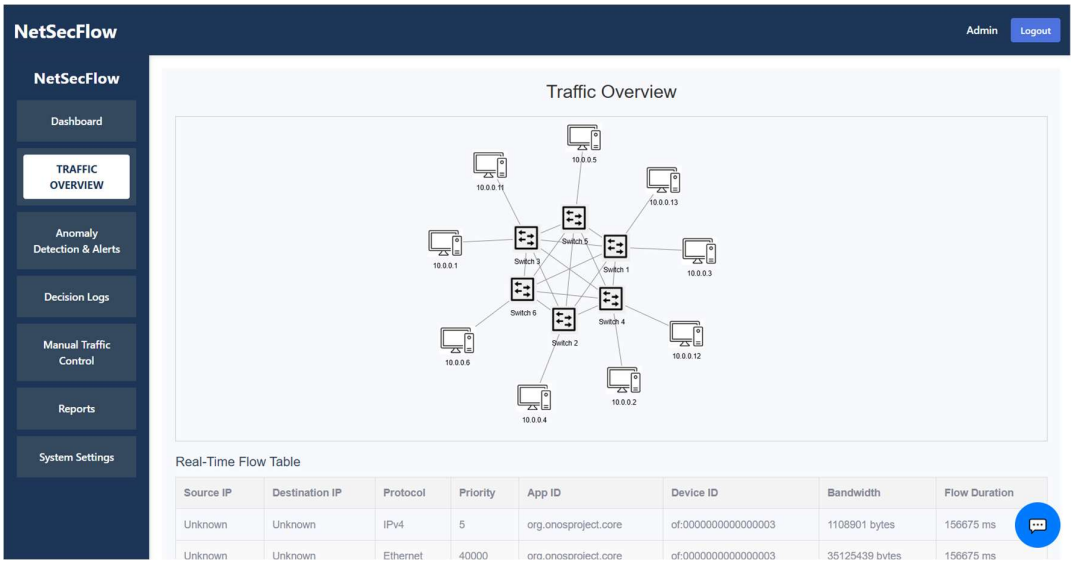


Figure 8 NetSecFlow Traffic Overview Page

This page presents detailed network topology, traffic analysis, highlighting patterns and anomalies to help administrators monitor network behavior effectively.

4.2.3 Anomaly detection & Alert Page

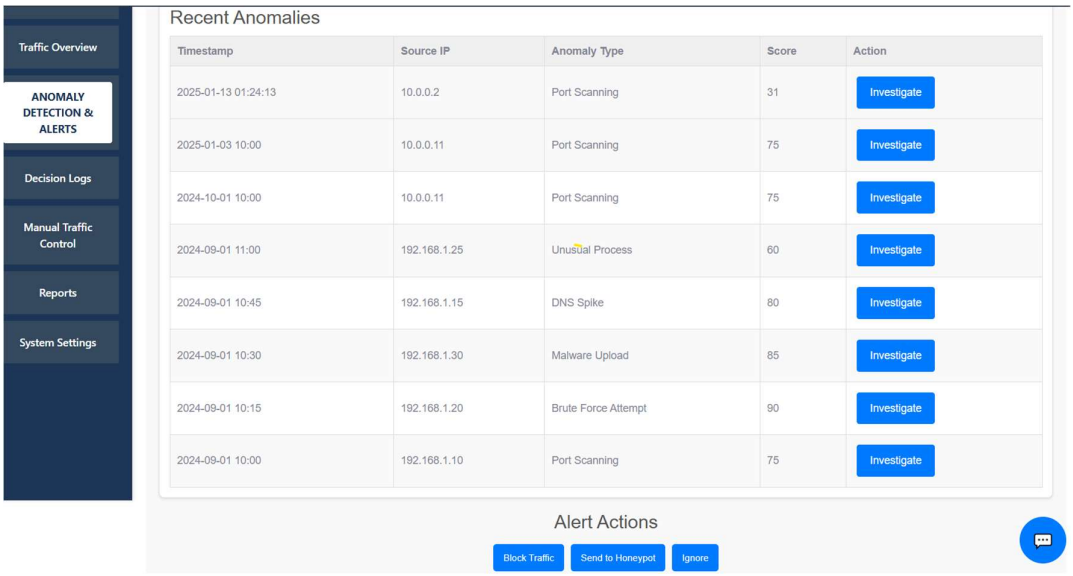


Figure 9 NetSecFlow Anomaly Detection Main Page

This page displays the detected anomalies, allowing administrators to review potential threats and take necessary actions.

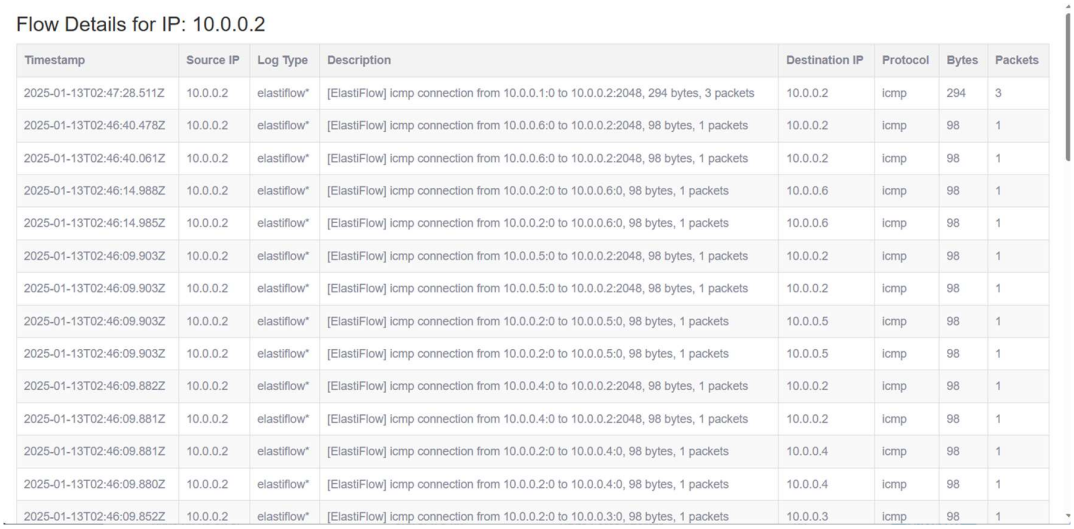


Figure 10 NetSecFlow Anomaly Detection Page(Investigate Button)

This page highlights the "Investigate" button functionality, allowing administrators to pull all events from ELK related to the source IP.

Figure 11 NetSecFlow Anomaly Detection Page(Block Button)

This page highlights the "Block" button functionality, allowing administrators to block malicious traffic directly from the dashboard.

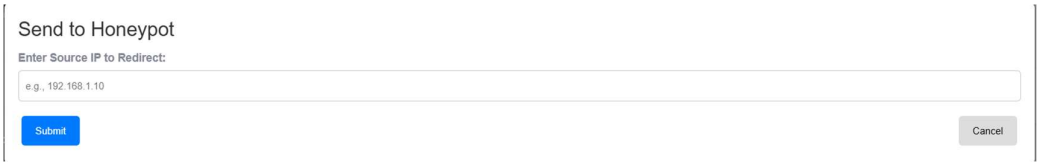


Figure 12 NetSecFlow Anomaly Detection Page(Send to Honeypot Button)

This page demonstrates the "Send to Honeypot" button, used to isolate potential threats for further analysis without impacting the main network.

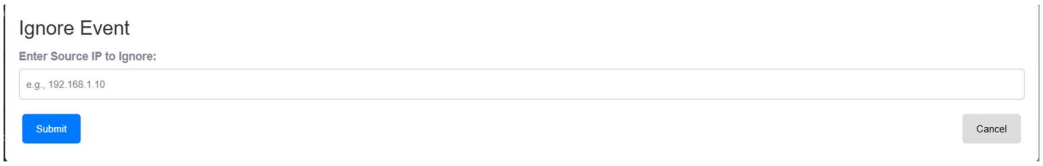


Figure 13 NetSecFlow Anomaly Detection Page(Send to Ignore Event Button)

This page features the "Ignore Event" button, enabling administrators to dismiss false positives or low-risk events from the anomaly list.

4.2.4 Decision Logs Page

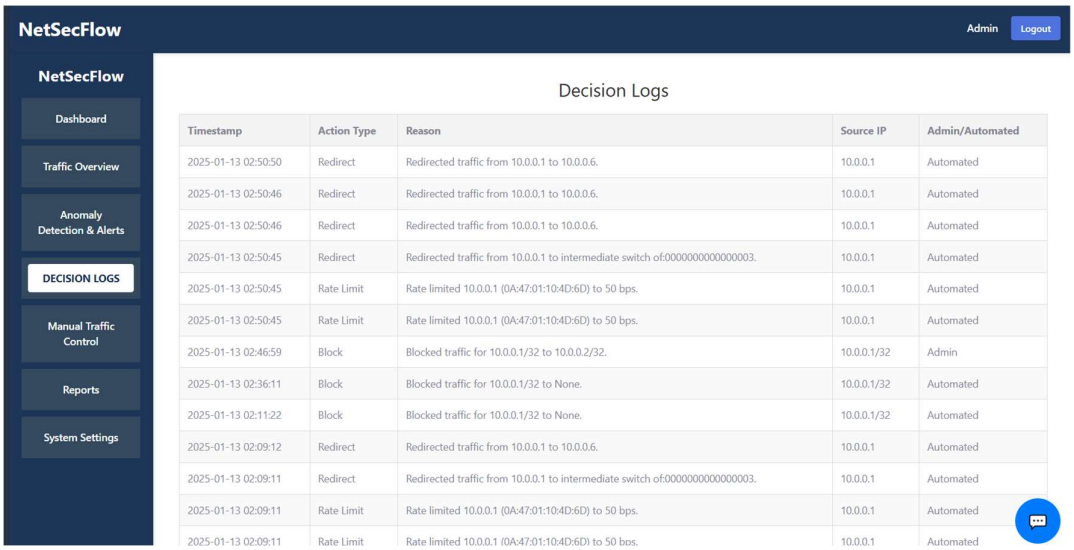


Figure 14 NetSecFlow Decision Logs Page

This page records all automated and manual actions taken by the system, providing a complete log for auditing and future investigations.

4.2.5 Manual Traffic Control Page

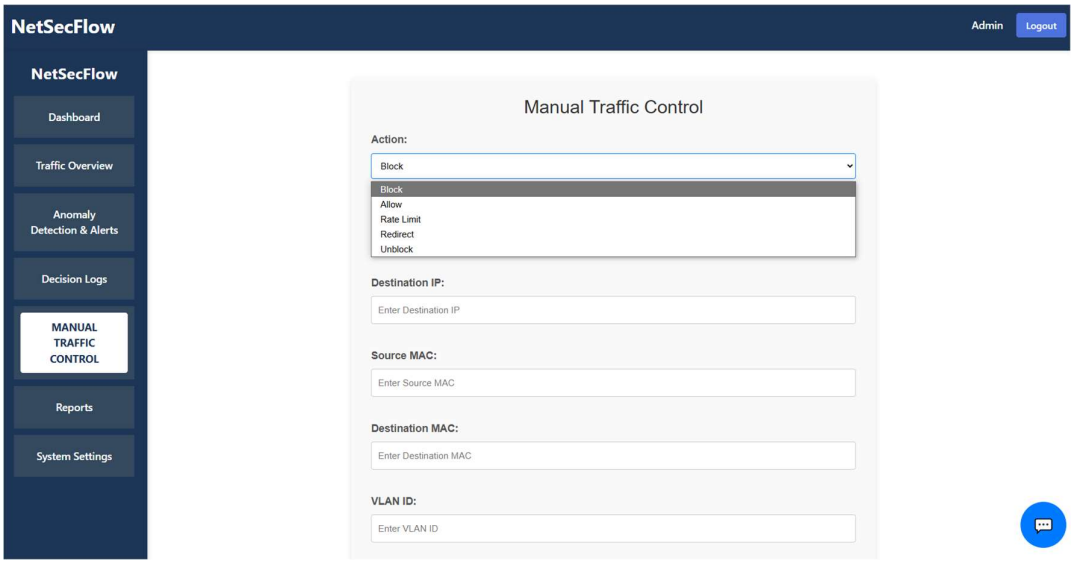


Figure 15 NetSecFlow Traffic Control Page(Block)

This page enables manual blocking of traffic based on user-defined criteria, such as source or destination IP and MAC addresses.

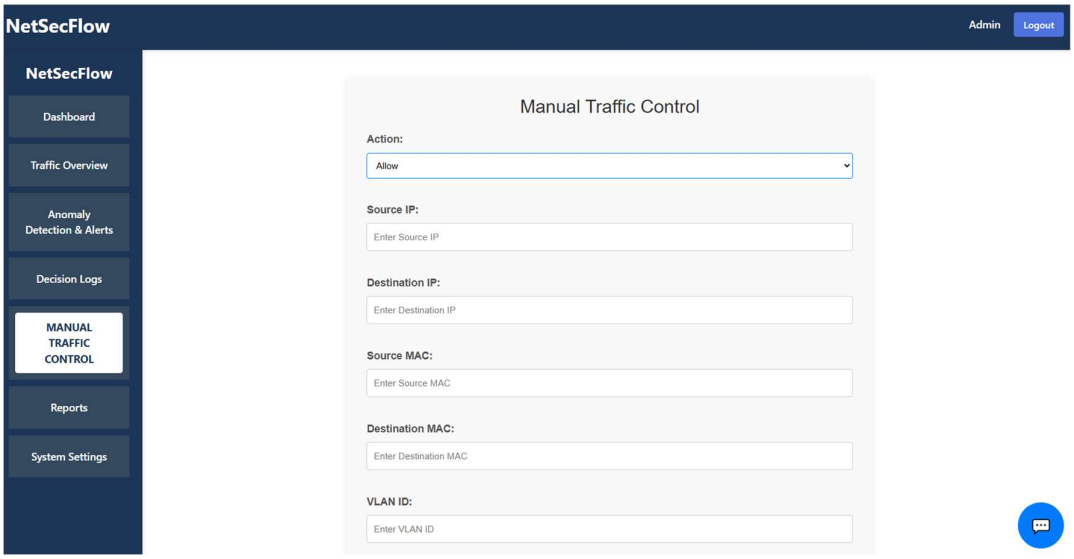


Figure 16 NetSecFlow Traffic Control Page(Allow)

This page allows administrators to explicitly permit traffic by creating rules to override automated decisions. It has higher priority than the Block Rule.

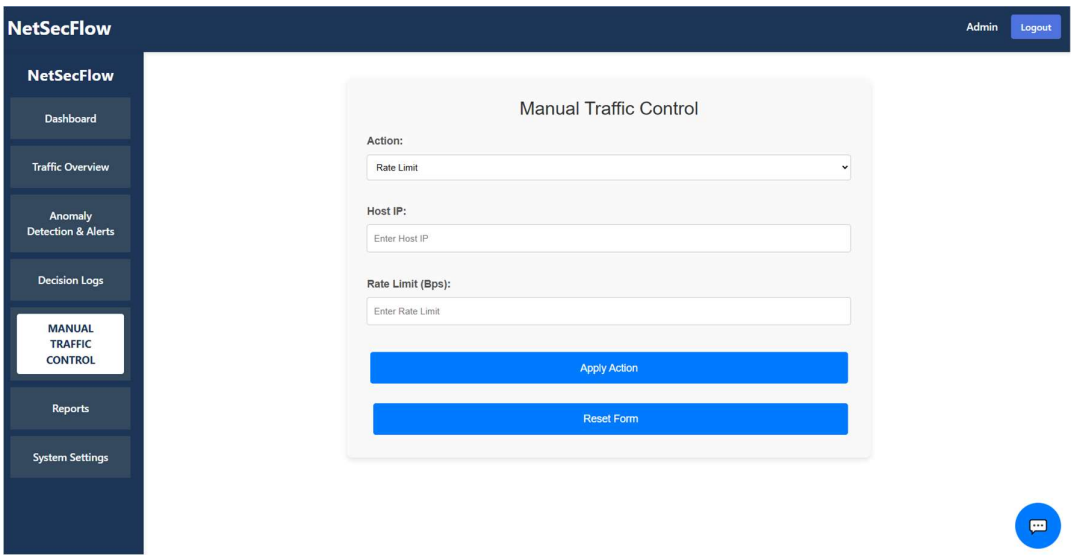


Figure 17 NetSecFlow Traffic Control Page(Rate Limit)

This page lets administrators impose rate limits on specific traffic to prevent overloading the network while maintaining partial connectivity.

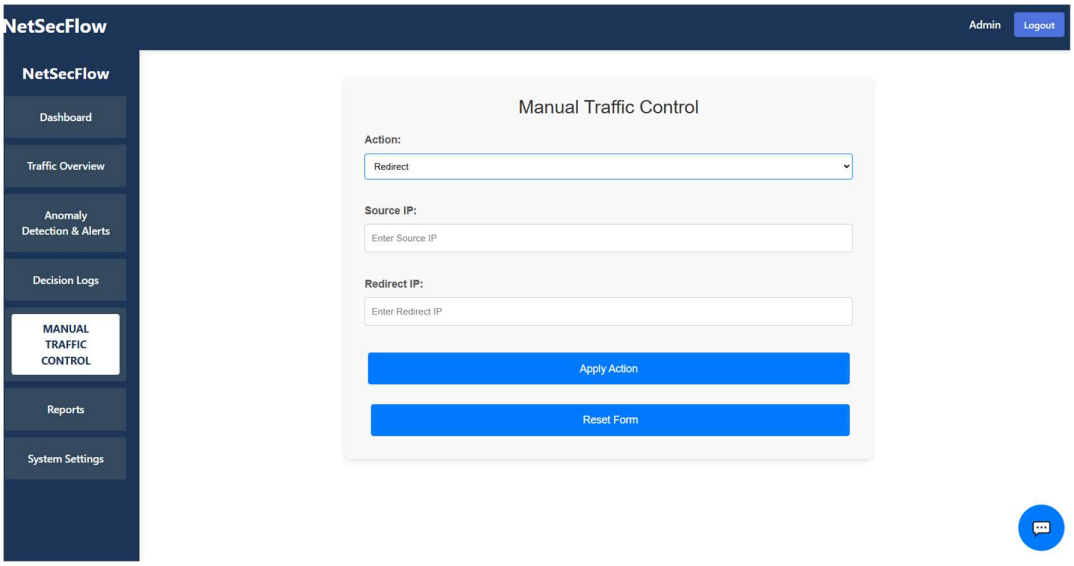


Figure 18 NetSecFlow Traffic Control Page(Redirect)

This page provides the option to redirect traffic to alternative paths or honeypots for threat isolation or analysis.

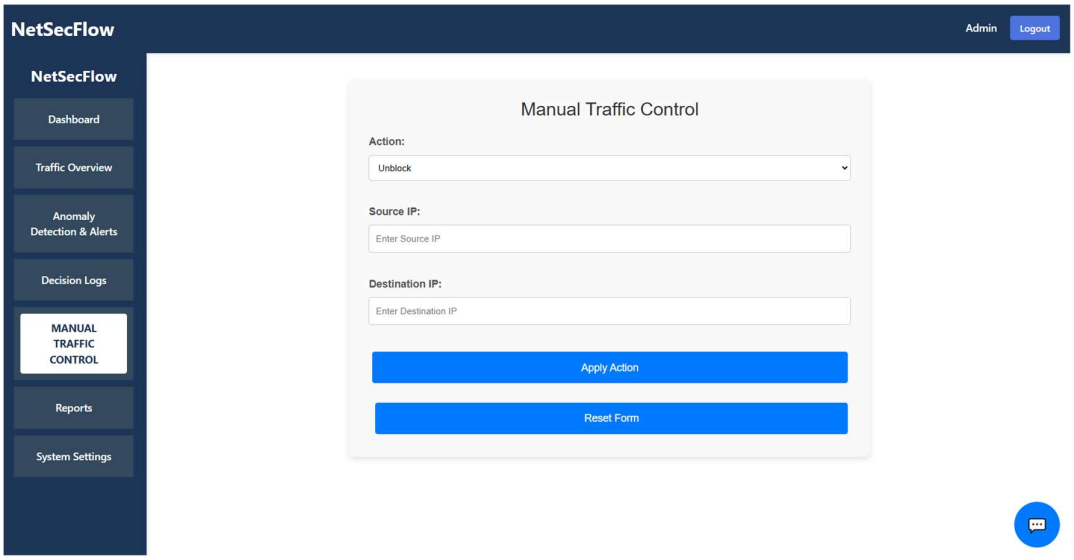


Figure 19 NetSecFlow Traffic Control Page(Unblock)

This page allows administrators to lift previously applied blocks, restoring traffic flow to its original state.

4.2.6 Reports Page

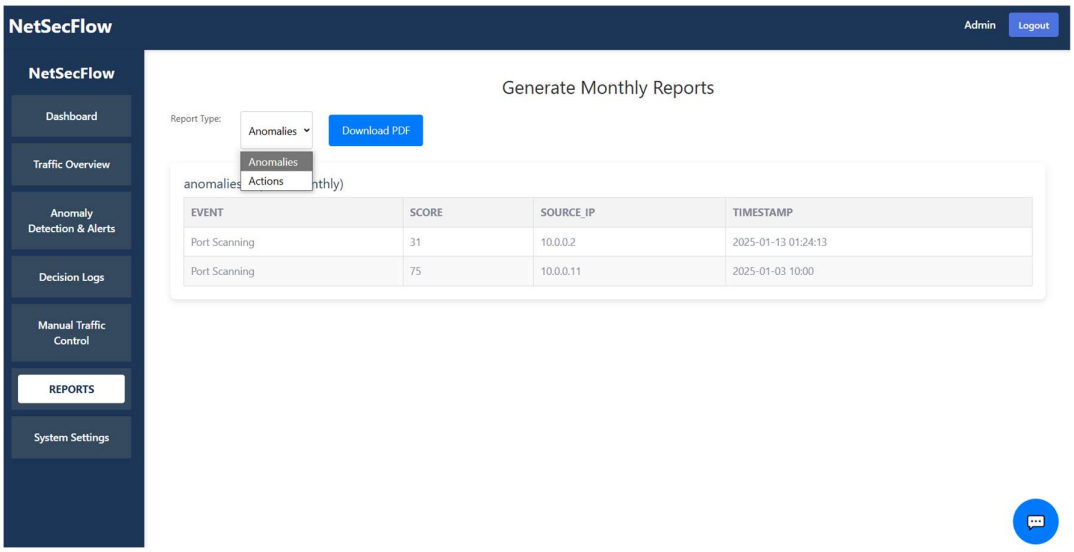


Figure 20 NetSecFlow Reports Page

This page generates detailed reports on anomalies and actions taken, providing valuable insights for administrators. A pdf report will be generated.

4.2.7 System Settings Page

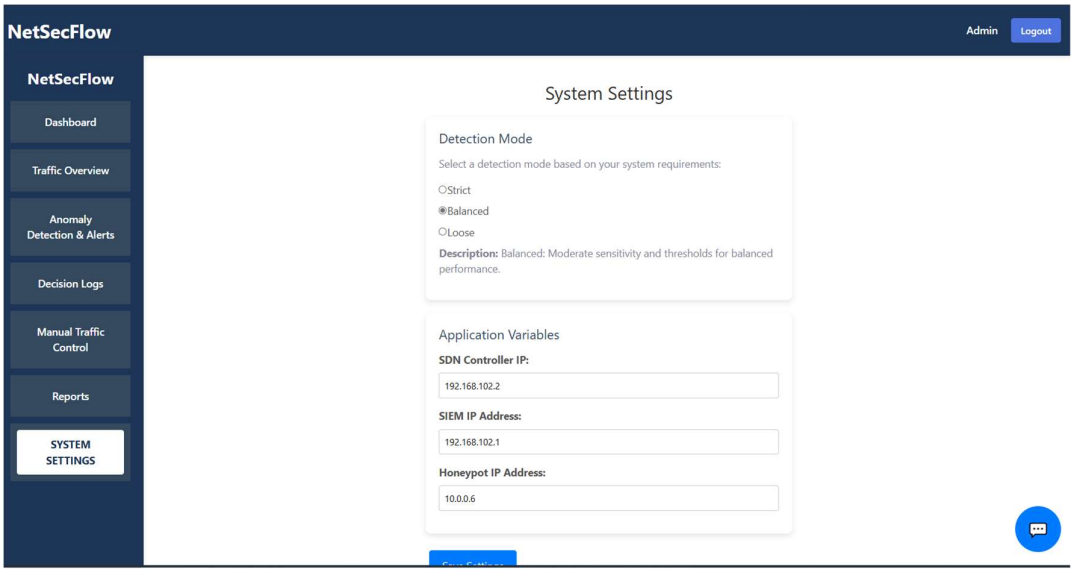


Figure 21 NetSecFlow Settings Page

This page lets administrators configure system settings, including SDN controller IPs, honeypot addresses, and thresholds for decision-making.

4.2.8 Chatbot Assistant

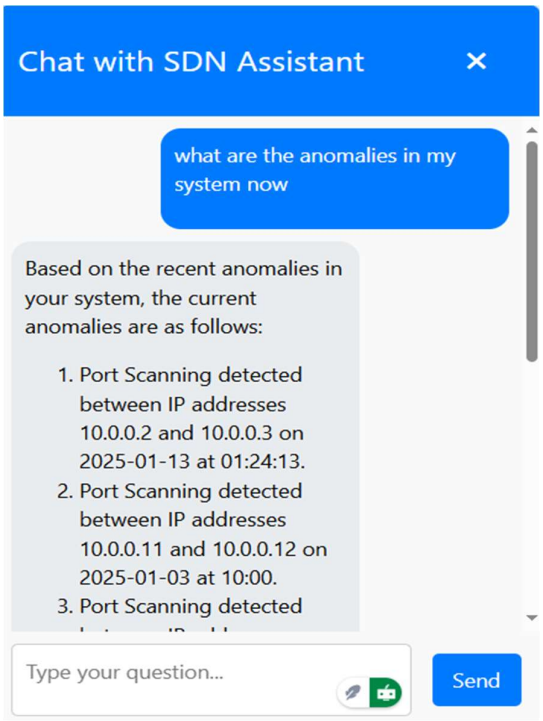


Figure 22 NetSecFlow Chatbot Assistant

This page introduces a chatbot assistant that helps administrators navigate the system and retrieve information through conversational queries.

4.2.9 ONOS Application

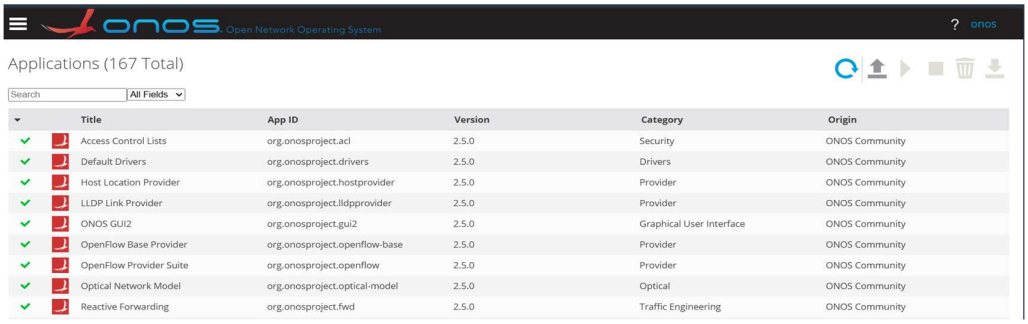


Figure 23 ONOS Controller Applications Page

This page displays the applications running on the ONOS SDN controller, showing their status and functionalities. These are the applications that I activated throughout the project for my project to function.

4.2.10 ONOS Clusters

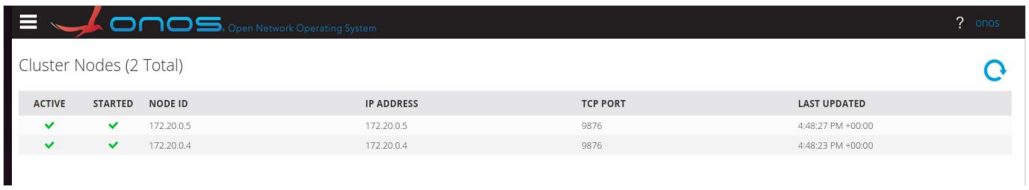


Figure 24 ONOS Controller Cluster Nodes Page

This page presents the cluster nodes in the ONOS controller, providing insights into the system's distributed architecture and fault tolerance. Currently it shows that the two controller nodes are deployed in docker mode and sync each other to provide tolerance.

4.2.11 ONOS Flows

Figure 25 ONOS Controller Flows Page

This page shows the flow rules managed by the ONOS controller, enabling administrators to monitor and troubleshoot traffic flows.

4.2.12 ONOS Intent

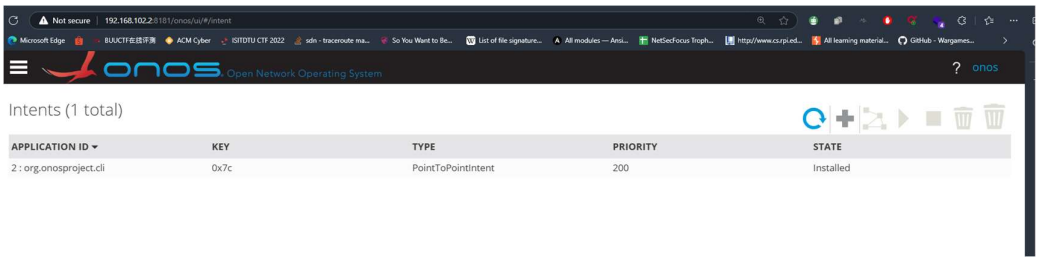


Figure 26 ONOS Controller Intent Page

This page lists the intents configured in the ONOS controller, showcasing traffic engineering and routing rules applied in the network.

4.2.13 ELK Index Management

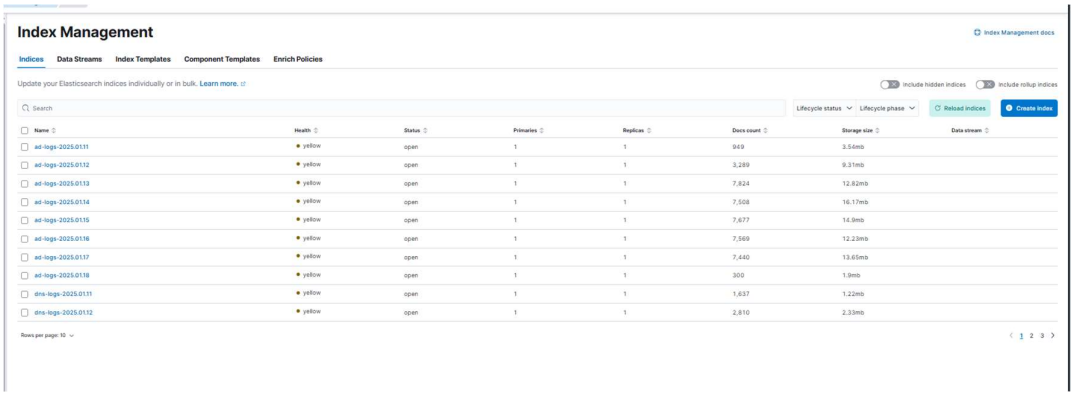


Figure 27 Kibana Index Management Page

This page displays the index management features of Kibana, allowing administrators to manage and optimize Elasticsearch indices. There are different indices collected like ad-logs, dns-logs, cowrie, elastiflow, suricata and wazuh.

4.2.14 ELK Kibana Dashboard

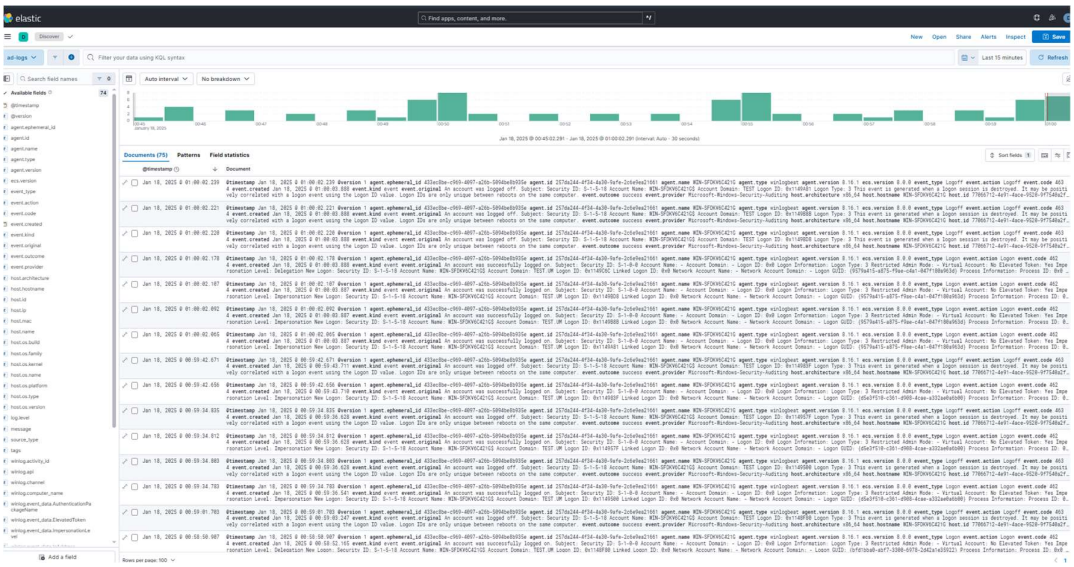


Figure 28 Kibana Index Discover Page

This page provides a discovery view for exploring indexed data, enabling detailed analysis of logs and events captured by the system.

4.2.15 ELK Rules

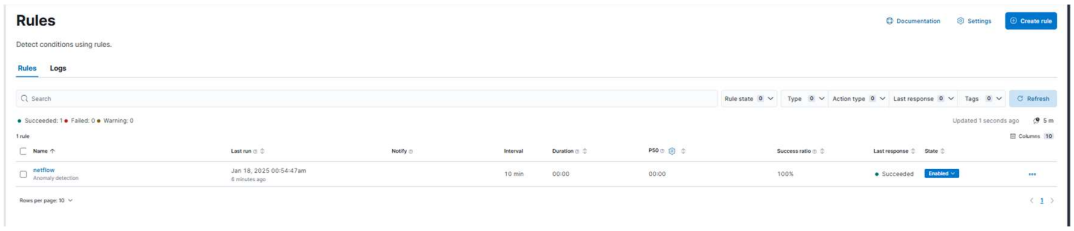


Figure 29 Kibana Rules Page

This page shows the alerting rules configured in Kibana, allowing administrators to define and monitor thresholds for triggering actions.

4.2.16 ELK Connectors

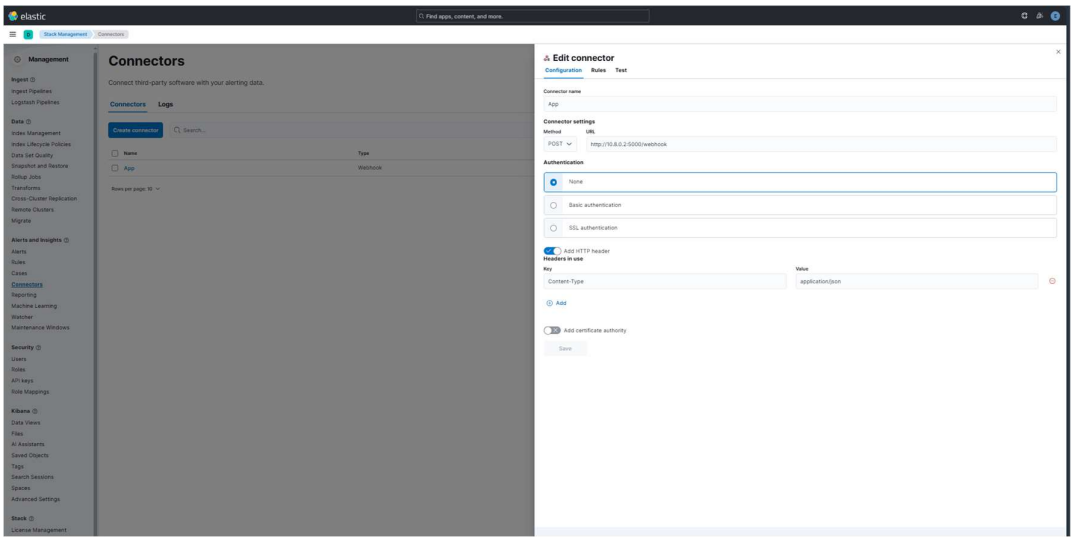


Figure 30 Kibana Connectors Page

This page manages connectors in Kibana, integrating external systems like webhooks for alerting and action automation.

4.3 NetSecFlow Deployment

Deploying the NetSecFlow system involves setting up both the frontend and backend components, configuring the necessary dependencies, and ensuring integration with essential external systems like the SDN controller (ONOS) and SIEM tools (Elasticsearch and Kibana). This guide provides step-by-step instructions to get the system running seamlessly.

```
Microsoft Windows [Version 10.0.19045.5371]
(c) Microsoft Corporation. All rights reserved.

C:\Users\jiaya>git clone https://github.com/heheJY/netsecflow.git
```

Figure 31 Git Clone Application Screenshots

To begin the deployment process, the first step is to clone the NetSecFlow repository from GitHub to your local machine. By cloning the repository, you will have access to all the required files, including the frontend, backend, and database configurations. Use the

git clone command, followed by navigating to the project directory using cd netsecflow. This will set up the base project structure for further configuration.

Figure 32 React Run Screenshot

The frontend of the system, developed using React.js, provides the user interface for monitoring and controlling network traffic. To deploy the frontend, navigate to the frontend directory and install the required Node.js modules by running npm install. This ensures that all dependencies are correctly configured. Once the dependencies are installed, you can start the development server using npm start. The application will then be accessible in your browser at http://localhost:3000. For production deployments, you can create a production-ready build using the command npm run build, which generates optimized files in the build/ directory.

Figure 33 Python Flask Run Screenshot

The backend of NetSecFlow, built using Python and Flask, manages the core system logic, including APIs for traffic control actions, integration with ONOS and ELK, and data processing. To deploy the backend, navigate to the backend directory and set up a Python virtual environment to isolate dependencies. Activate the virtual environment and install the required libraries listed in the requirements.txt file using the pip install command. Once the dependencies are installed, configure the database. For local

development, SQLite can be used, and you can initialize the database schema by calling the init_db() function in database.py. Finally, start the Flask application with the command python run.py, making the backend accessible at http://localhost:5000.

In addition to the frontend and backend, the NetSecFlow system requires integration with the ONOS SDN controller to manage network flows. Ensure that the ONOS controller is installed and running, with the necessary network topology configured. Update the backend settings to include the correct IP address of the SDN controller. Similarly, Elasticsearch and Kibana serve as the system's SIEM tools for anomaly detection and threat analysis. Configure these tools by setting up indices and connectors in Elasticsearch, and update the backend settings with the correct Elasticsearch IP address to establish seamless integration.

Once both the frontend and backend are running, along with the ONOS controller and SIEM tools, you can access the complete NetSecFlow system. The frontend interface is available at http://localhost:3000, while the backend APIs can be accessed at http://localhost:5000. Ensure the backend and frontend are properly connected by verifying the API endpoints in the frontend configuration. By following these deployment instructions, NetSecFlow will be fully operational, ready to enhance network security and provide comprehensive traffic management capabilities.

4.4 NetSecFlow Implementation

This section will mention about the code implementation and logic for some important functions for the application including webhook, decision, and controller actions.

```python
@app.route('/webhook', methods=['POST'])
def webhook():
    data = request.get_json()
    if data:
        try:
            save_event(data)  # Save the received data to the database
            process_event(data)  # Process the data instantly
            return {'message': 'Event processed successfully'}, 200
        except Exception as e:
            print(f"Error processing event: {e}")
            return {'message': 'Failed to process event', 'error': str(e)}, 500
    return {'message': 'No data received'}, 400
```

Figure 34 Webhook in reception.py

The provided webhook function is a Flask endpoint designed to handle incoming HTTP POST requests. It acts as a key component of the system by processing event data received from external sources, such as alerts from a monitoring tool or SIEM system. When the function receives a JSON payload via the request.get_json() method, it first checks if the data is valid.

If valid data is received, the function attempts to execute two main operations: **saving the event** and **processing the event**. The save_event function is responsible for persisting the event data into the database, ensuring that a historical record of all events is maintained for auditing or analysis purposes. It typically extracts relevant fields, such as source IP, destination IP, event type, timestamp, and any additional details, and saves them in a structured format.

The second operation, process_event, analyzes the data to take further actions based on its content. This step might involve anomaly detection using predefined rules or machine learning models. For example, if the event data indicates suspicious activity, the

system might trigger a predefined action, such as blocking a source IP, sending an alert, or redirecting traffic to a honeypot.

```
confidence_rate = CONFIDENCE_RATES[source]
detection_multiplier = DETECTION_MODES[detection_mode]

# Calculate marks increase
marks_increase = score * confidence_rate * detection_multiplier

# Update marks for SourceIP
marks = get_or_initialize_marks(source_ip)
updated_marks = marks + marks_increase
save_marks(source_ip, updated_marks)

actions = []  # List to collect actions for informational purposes

# Take actions based on marks thresholds
if updated_marks >= 100:
    block_ip(src_ip=f"{source_ip}/32", admin_or_automated="Automated")
    actions.append("Block")
elif updated_marks >= 60:
    redirect_traffic_full(source_ip, honeypot_ip, admin_or_automated="Automated")
    actions.append("Redirect")
elif updated_marks >= 40:
    rate_limit_for_host(
        source_ip,
        rate_limit_bps=50,
        admin_or_automated="Automated"  # Mark the action as automated
    )
    actions.append("Rate Limit")
elif updated_marks >= 20:
    redirect_to_intermediate_ovs(source_ip, ovs_id="of:0000000000000003", admin_or_automated="Automated")
    actions.append("Intermediate Redirect")
else:
    actions.append("No Action")

return actions
```

Figure 35 Decision Logic in decision.py

This code handles decision-making based on a scoring system to determine network actions for a given source_ip. It calculates the marks_increase using a confidence rate and detection multiplier derived from predefined constants. The current marks for the source_ip are updated, saved, and used to decide an appropriate action based on thresholds. This is part of the process_event.

```python
def block_ip(src_ip=None, dst_ip=None, src_mac=None, dst_mac=None, vlan_id=None, eth_type=None, ip_proto=None, src_port=None, dst_port=None, admin_or_automated="Admin"):
    log_action(
        action_type="Block",
        reason=f"Blocked traffic for {src_ip} to {dst_ip}.",
        source_ip=src_ip,
        admin_or_automated=admin_or_automated
    )
    create_acl_rule(
        src_ip=src_ip, dst_ip=dst_ip, src_mac=src_mac, dst_mac=dst_mac,
        vlan_id=vlan_id, eth_type=eth_type, src_port=src_port, dst_port=dst_port, ip_proto=ip_proto,
        action="DENY"
    )
```

```python
def allow_ip(src_ip=None, dst_ip=None, src_mac=None, dst_mac=None, vlan_id=None, eth_type=None, ip_proto=None, src_port=None, dst_port=None, admin_or_automated="Admin"):
    log_action(
        action_type="Allow",
        reason=f"Allow traffic from {src_ip} to {dst_ip}.",
        source_ip=src_ip,
        admin_or_automated=admin_or_automated
    )
    create_acl_rule(
        src_ip=src_ip, dst_ip=dst_ip, src_mac=src_mac, dst_mac=dst_mac,
        vlan_id=vlan_id, eth_type=eth_type, src_port=src_port, dst_port=dst_port, ip_proto=ip_proto,
        action="ALLOW"
    )
```

Figure 36 Block and Allow action in controller_actions.py

The block and allow action is based on ACL application in ONOS controller, we can create the rule and apply in on controller, so it can helps to block or allow a traffic.

```python
def unblock_ip(src_ip=None, dst_ip=None, src_port=None, dst_port=None, prefix_length='32',admin_or_automated="Admin"):
    acl_rules = list_acl_rules()
    log_action(
        action_type="Unblock",
        reason=f"Unblock traffic from {src_ip} to {dst_ip}.",
        source_ip=src_ip,
        admin_or_automated=admin_or_automated
    )
    if src_ip and not is_valid_ip(src_ip):
        print(f"Invalid source IP address: {src_ip}")
        return
    if dst_ip and not is_valid_ip(dst_ip):
        print(f"Invalid destination IP address: {dst_ip}")
        return

    if src_ip and '/' not in src_ip:
        src_ip = f"{src_ip}/{prefix_length}"
    if dst_ip and '/' not in dst_ip:
        dst_ip = f"{dst_ip}/{prefix_length}"

    for rule in acl_rules:
        if isinstance(rule, dict):
            rule_src_ip = rule.get("srcIp", "").strip()
            rule_dst_ip = rule.get("dstIp", "").strip()

            if ((src_ip and rule_src_ip == src_ip) or (dst_ip and rule_dst_ip == dst_ip)):
                rule_id = rule.get("id")
                delete_url = f"{ONOS_API_BASE}/acl/rules/{rule_id}"
                try:
                    response = requests.delete(delete_url, auth=auth)
                    if response.status_code == 204:
                        print(f"Successfully unblocked traffic for IP: {src_ip or dst_ip}")
                    else:
                        print(f"Failed to delete ACL rule for IP {src_ip or dst_ip}: {response.text}")
                except requests.exceptions.RequestException as e:
                    print(f"Error deleting ACL rule: {e}")
    print(f"No ACL rule found for IP {src_ip or dst_ip}")
```

Figure 37 Unblock in controller_actions.py

This unblock function is listing down all the “BLOCK ACTION” acl rules in the onos controller and use DELETE in http request to remove for the acl rule implemented.

```
# Create a meter for rate limiting
meter_id = str(int(time.time()))  # Generate a unique meter ID
meter_config = {
    "deviceId": device_id,
    "meterId": meter_id,
    "unit": "KB_PER_SEC",  # Unit for the rate limit
    "bands": [
        {
            "type": "DROP",
            "rate": rate_limit_bps,
            "burstSize": 1000  # Optional burst size
        }
    ]
}

# Create the meter on the switch
print("Creating meter on the device...")
meter_url = f"{ONOS_API_BASE}/meters/{device_id}"
meter_response = requests.post(meter_url, json=meter_config, auth=auth)
if meter_response.status_code != 201:
    print(f"Failed to create meter: {meter_response.text}")
    return

# Debugging: Fetch all meters to confirm creation
print("Fetching all meters after creation...")
meters_response = requests.get(meter_url, auth=auth)
if meters_response.status_code == 200:
    meters = meters_response.json().get("meters", [])
    print("Meters fetched:", meters)

    # Find the most recent meter for debugging
    latest_meter = next((m for m in meters if m["bands"][0]["rate"] == rate_limit_bps), None)
    if not latest_meter:
        print("Could not find the newly created meter.")
        return
    meter_id = latest_meter["id"]

    # Create a flow with MAC-based selector
    print("Attaching meter to flow...")
    flow_rule = {
        "priority": 41000,
        "timeout": 0,
        "isPermanent": True,
```

```
# Create a flow with MAC-based selector
print("Attaching meter to flow...")
flow_rule = {
    "priority": 41000,
    "timeout": 0,
    "isPermanent": True,
    "deviceId": device_id,
    "treatment": {
        "instructions": [
            {"type": "METER", "meterId": meter_id},
            {"type": "OUTPUT", "port": inbound_port}
        ]
    },
    "selector": {
        "criteria": [
            {"type": "ETH_TYPE", "ethType": "0x0800"},  # IPv4 Traffic
            {"type": "ETH_DST", "mac": host_mac},  # Match Source MAC
        ]
    }
}

flow_url = f"{ONOS_API_BASE}/flows/{device_id}"
flow_response = requests.post(flow_url, json=flow_rule, auth=auth)
```

Figure 38 RateLimiting in controller_actions.py

A unique meter ID is generated based on the current time. The meter configuration specifies the rate limit (in bits per second), along with actions to take when traffic exceeds the limit (e.g., dropping excess traffic). This configuration is sent to the ONOS controller using a POST request. If the meter creation fails, the process stops, and an error is logged.

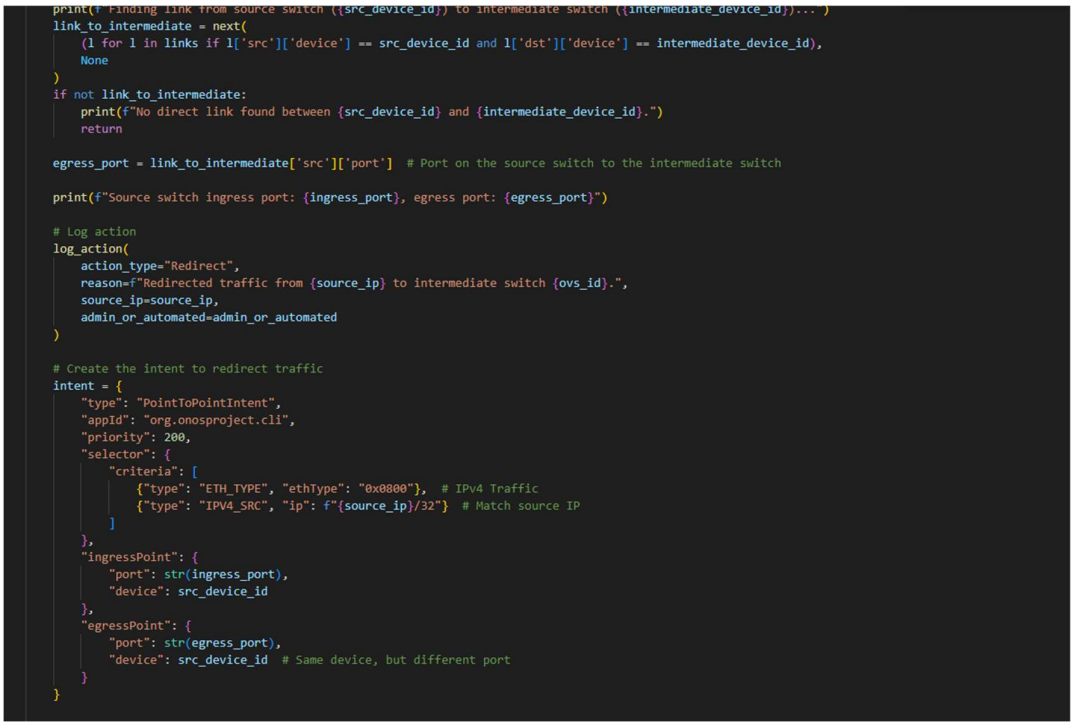
A flow rule is created and configured to match specific traffic criteria, such as IPv4 packets (ETH_TYPE: 0x0800) and traffic originating from a specific MAC address (ETH_DST). The flow rule links the created meter to enforce rate limiting and specifies the output port where the traffic should be forwarded. This flow rule is sent to ONOS via a POST request.

```
# Locate source host
print(f"Finding source host with IP {source_ip}...")
source_host = next((host for host in hosts if source_ip in host.get('ipAddresses', [])), None)
if not source_host:
    print(f"Source host with IP {source_ip} not found.")
    return

# Locate honeypot host
print(f"Finding honeypot host with IP {honeypot_ip}...")
honeypot_host = next((host for host in hosts if honeypot_ip in host.get('ipAddresses', [])), None)
if not honeypot_host:
    print(f"Honeypot host with IP {honeypot_ip} not found.")
    return

# Extract source and destination device details
src_location = source_host['locations'][0]
dst_location = honeypot_host['locations'][0]

src_device_id = src_location['elementId']
src_port = src_location['port']
dst_device_id = dst_location['elementId']
dst_port = dst_location['port']

# Determine shortest path
print(f"Calculating the shortest path from {src_device_id} to {dst_device_id}...")
path = calculate_shortest_path(src_device_id, dst_device_id, links)
if not path:
    print("No path found between the source and honeypot.")
    return

print(f"Shortest path calculated: {path}")

# Generate the Point-to-Point Intent JSON
create_point_to_point_intents(
    source_ip=source_ip,
    destination_ip=honeypot_ip,
    src_device=src_device_id,
    dst_device=dst_device_id,
    src_port=src_port,
    dst_port=dst_port
)
```

Figure 39 Redirect to Honeypot in controller_actions.py

The code searches for the source host using its IP address from the topology data. If the source host is not found, an error is logged, and the process stops. This ensures that the redirection logic only proceeds if the source host is identified in the network. Once the source and honeypot hosts are located, their respective device IDs (the network switches they are connected to) and ports are extracted from the topology data. These details are used to construct the Point-to-Point Intent. The code calculates the shortest path between the source and honeypot devices using the network's link data. If no path is

found, an error is logged, and the process stops. This ensures the redirection follows the optimal route.



```
print(f"Finding link from source switch ({src_device_id}) to intermediate switch ({intermediate_device_id})...")
link_to_intermediate = next(
    (l for l in links if l['src']['device'] == src_device_id and l['dst']['device'] == intermediate_device_id),
    None
)
if not link_to_intermediate:
    print(f"No direct link found between {src_device_id} and {intermediate_device_id}.")
    return

egress_port = link_to_intermediate['src']['port']  # Port on the source switch to the intermediate switch

print(f"Source switch ingress port: {ingress_port}, egress port: {egress_port}")

# Log action
log_action(
    action_type="Redirect",
    reason=f"Redirected traffic from {source_ip} to intermediate switch {ovs_id}.",
    source_ip=source_ip,
    admin_or_automated=admin_or_automated
)

# Create the intent to redirect traffic
intent = {
    "type": "PointToPointIntent",
    "appId": "org.onosproject.cli",
    "priority": 200,
    "selector": {
        "criteria": [
            {"type": "ETH_TYPE", "ethType": "0x0800"},  # IPv4 Traffic
            {"type": "IPV4_SRC", "ip": f"{source_ip}/32"}  # Match source IP
        ]
    },
    "ingressPoint": {
        "port": str(ingress_port),
        "device": src_device_id
    },
    "egressPoint": {
        "port": str(egress_port),
        "device": src_device_id  # Same device, but different port
    }
}
```

Figure 40 Reroute to intermediate switch in controller_actions.py

The code identifies the link between the source switch (src_device_id) and the intermediate switch (intermediate_device_id). If no direct link is found, an error message is logged, and the process is terminated. This ensures that rerouting is only attempted when a valid link exists between the devices. The ingress and egress ports are extracted from the link details. The ingress port is on the source switch, and the egress port is on the same switch, pointing towards the intermediate switch. These ports are essential for defining the rerouting path.

4.5 NetSecFlow Evaluation

4.5.1 System Functionality Tests

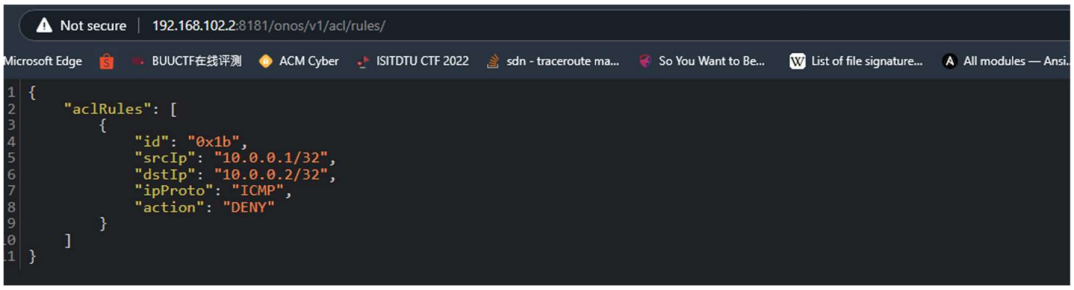
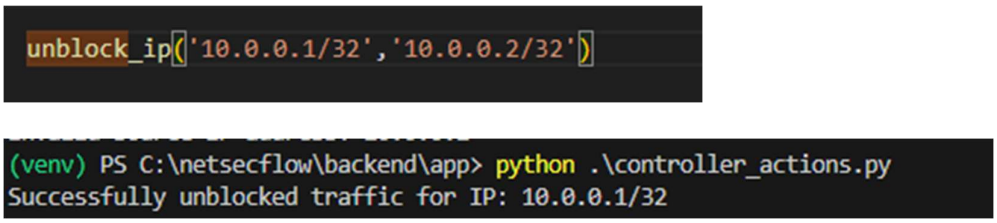


Figure 41 Block Testing(Similar Logic Apply to Allow)



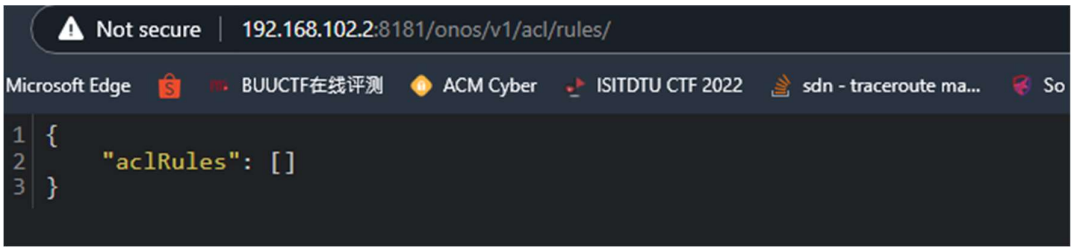


Figure 42 Unblock Testing

```
def test_rate_limit_for_host():
    """
    Test applying rate limiting on the closest inbound port of a host.
    """
    print("\n--- Testing Rate Limiting for Host ---")
    host_ip = "10.0.0.1"
    rate_limit_bps = 50

    try:
        rate_limit_for_host(
            host_ip=host_ip,
            rate_limit_bps=rate_limit_bps,
            admin_or_automated="Automated"  # Mark the action as automated
        )
        print(f"Rate limiting test completed for host: {host_ip}")
    except Exception as e:
        print(f"Error during rate limiting test: {e}")
```

```
(venv) PS C:\netsecflow\backend\app> python .\controller_actions.py

--- Testing Rate Limiting for Host ---
Fetching topology data...
Searching for host with IP 10.0.0.1...
Creating meter on the device...
Fetching all meters after creation...
Meters fetched: [{'id': '5', 'life': 566925, 'packets': 26, 'bytes': 2548, 'referenceCount': 1, 'unit': 'KB_PER_SEC', 'burst': False, 'deviceId': 'of:0000000000000001', 'appId': 'org.ono
sproject.rest', 'state': 'ADDED', 'bands': [{'type': 'DROP', 'rate': 50, 'packets': 0, 'bytes': 0, 'burstSize': 1000}]}, {'id': '6', 'life': 566314, 'packets': 0, 'bytes': 0, 'referenceC
ount': 0, 'unit': 'KB_PER_SEC', 'burst': False, 'deviceId': 'of:0000000000000001', 'appId': 'org.onosproject.rest', 'state': 'ADDED', 'bands': [{'type': 'DROP', 'rate': 50, 'packets': 0,
 'bytes': 0, 'burstSize': 1000}]}, {'id': '7', 'life': 566313, 'packets': 0, 'bytes': 0, 'referenceCount': 0, 'unit': 'KB_PER_SEC', 'burst': False, 'deviceId': 'of:0000000000000001', 'ap
pId': 'org.onosproject.rest', 'state': 'ADDED', 'bands': [{'type': 'DROP', 'rate': 50, 'packets': 0, 'bytes': 0, 'burstSize': 1000}]}, {'id': '4', 'life': 566925, 'packets': 0, 'bytes':
0, 'referenceCount': 0, 'unit': 'KB_PER_SEC', 'burst': False, 'deviceId': 'of:0000000000000001', 'appId': 'org.onosproject.rest', 'state': 'ADDED', 'bands': [{'type': 'DROP', 'rate': 50,
 'packets': 0, 'bytes': 0, 'burstSize': 1000}]}, {'id': '9', 'life': 563819, 'packets': 0, 'bytes': 0, 'referenceCount': 0, 'unit': 'KB_PER_SEC', 'burst': False, 'deviceId': 'of:00000000
0000001', 'appId': 'org.onosproject.rest', 'state': 'ADDED', 'bands': [{'type': 'DROP', 'rate': 50, 'packets': 0, 'bytes': 0, 'burstSize': 1000}]}, {'id': '1', 'life': 619416, 'packets'
: 1070, 'bytes': 1611800, 'referenceCount': 0, 'unit': 'KB_PER_SEC', 'burst': False, 'deviceId': 'of:0000000000000001', 'appId': 'org.onosproject.rest', 'state': 'ADDED', 'bands': [{'typ
e': 'DROP', 'rate': 50, 'packets': 0, 'bytes': 0, 'burstSize': 1000}]}, {'id': 'a', 'life': 0, 'packets': 0, 'bytes': 0, 'referenceCount': 0, 'unit': 'KB_PER_SEC', 'burst': False, 'devic
eId': 'of:0000000000000001', 'appId': 'org.onosproject.rest', 'state': 'PENDING_ADD', 'bands': [{'type': 'DROP', 'rate': 50, 'packets': 0, 'bytes': 0, 'burstSize': 1000}]}, {'id': '2', '
life': 621591, 'packets': 0, 'bytes': 0, 'referenceCount': 0, 'unit': 'KB_PER_SEC', 'burst': False, 'deviceId': 'of:0000000000000001', 'appId': 'org.onosproject.rest', 'state': 'ADDED',
'bands': [{'type': 'DROP', 'rate': 125, 'packets': 0, 'bytes': 0, 'burstSize': 10}]}, {'id': '3', 'life': 566926, 'packets': 0, 'bytes': 0, 'referenceCount': 0, 'unit': 'KB_PER_SEC', 'bu
rst': False, 'deviceId': 'of:0000000000000001', 'appId': 'org.onosproject.rest', 'state': 'ADDED', 'bands': [{'type': 'DROP', 'rate': 50, 'packets': 0, 'bytes': 0, 'burstSize': 1000}]},
{'id': '8', 'life': 563820, 'packets': 0, 'bytes': 0, 'referenceCount': 0, 'unit': 'KB_PER_SEC', 'burst': False, 'deviceId': 'of:0000000000000001', 'appId': 'org.onosproject.rest', 'stat
e': 'ADDED', 'bands': [{'type': 'DROP', 'rate': 50, 'packets': 0, 'bytes': 0, 'burstSize': 1000}]}]
Attaching meter to flow...
Successfully applied rate limit for 10.0.0.1 (0A:47:01:10:4D:6D) on port 1.
Rate limiting test completed for host: 10.0.0.1
```


```
{
    "meters": [
        {
            "id": "5",
            "life": 567167,
            "packets": 26,
            "bytes": 2548,
            "referenceCount": 1,
            "unit": "KB_PER_SEC",
            "burst": false,
            "deviceId": "of:0000000000000001",
            "appId": "org.onosproject.rest",
            "state": "ADDED",
            "bands": [
                {
                    "type": "DROP",
                    "rate": 50,
                    "packets": 0,
                    "bytes": 0,
                    "burstSize": 1000
                }
            ]
        },
        {
```

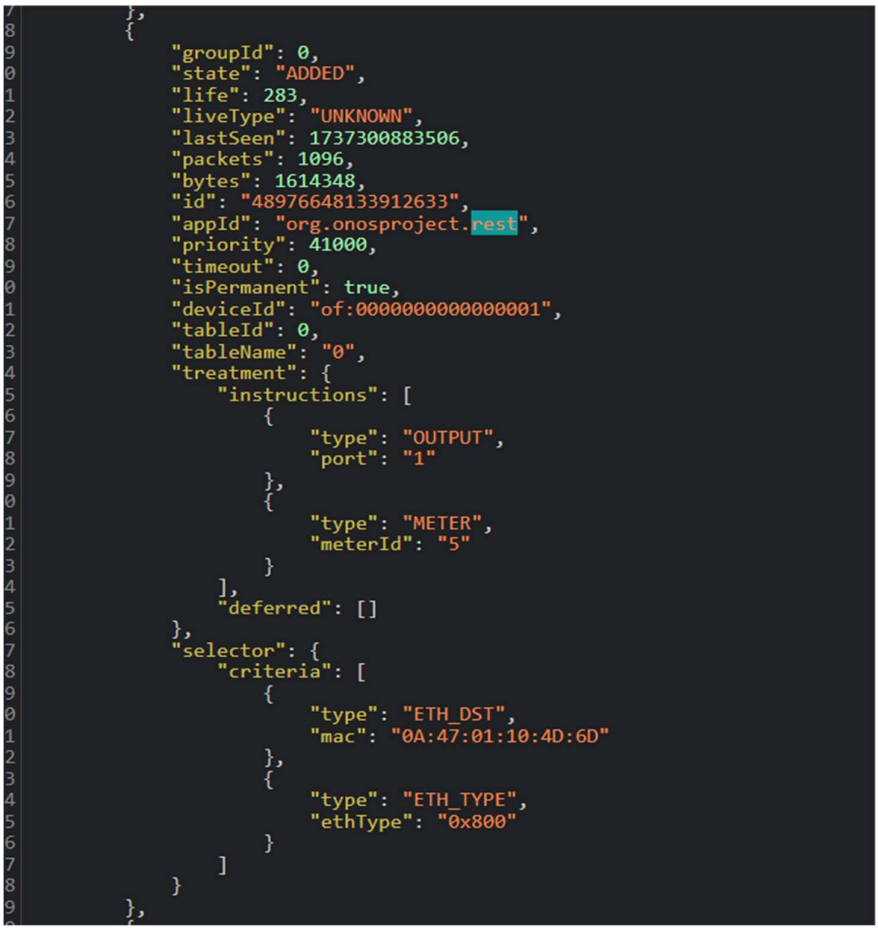
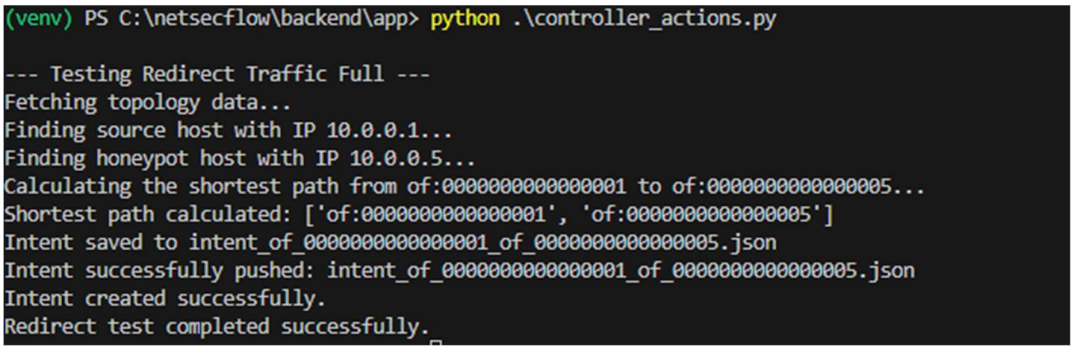


Figure 43 Rate Limit Testing



```
{
    "groupId": 0,
    "state": "ADDED",
    "life": 70,
    "liveType": "UNKNOWN",
    "lastSeen": 1737301033506,
    "packets": 0,
    "bytes": 0,
    "id": "48132224711170696",
    "appId": "org.onosproject.net.intent",
    "priority": 200,
    "timeout": 0,
    "isPermanent": true,
    "deviceId": "of:0000000000000001",
    "tableId": 0,
    "tableName": "0",
    "treatment": {
        "instructions": [
            {
                "type": "OUTPUT",
                "port": "5"
            }
        ],
        "deferred": []
    },
    "selector": {
        "criteria": [
            {
                "type": "IN_PORT",
                "port": 1
            },
            {
                "type": "ETH_TYPE",
                "ethType": "0x800"
            },
            {
                "type": "IPV4_SRC",
                "ip": "10.0.0.1/32"
            }
        ]
    }
},
```


Figure 44 Test Redirection

```
(venv) PS C:\netsecflow\backend\app> python .\controller_actions.py
Fetching topology data...
Finding source host with IP 10.0.0.1...
Finding intermediate Open vSwitch with ID of:0000000000000003...
Finding link from source switch (of:0000000000000001) to intermediate switch (of:0000000000000003)...
Source switch ingress port: 1, egress port: 3
Intent saved to intent_of_0000000000000001_of_0000000000000003.json
Successfully redirected traffic from 10.0.0.1 to of:0000000000000003.
```

Intents (2 total)

APPLICATION ID	KEY	TYPE	PRIORITY	STATE
2 : org.onosproject.cli	0x7f	PointToPointIntent	200	Installed
2 : org.onosproject.cli	0x82	PointToPointIntent	200	Installed

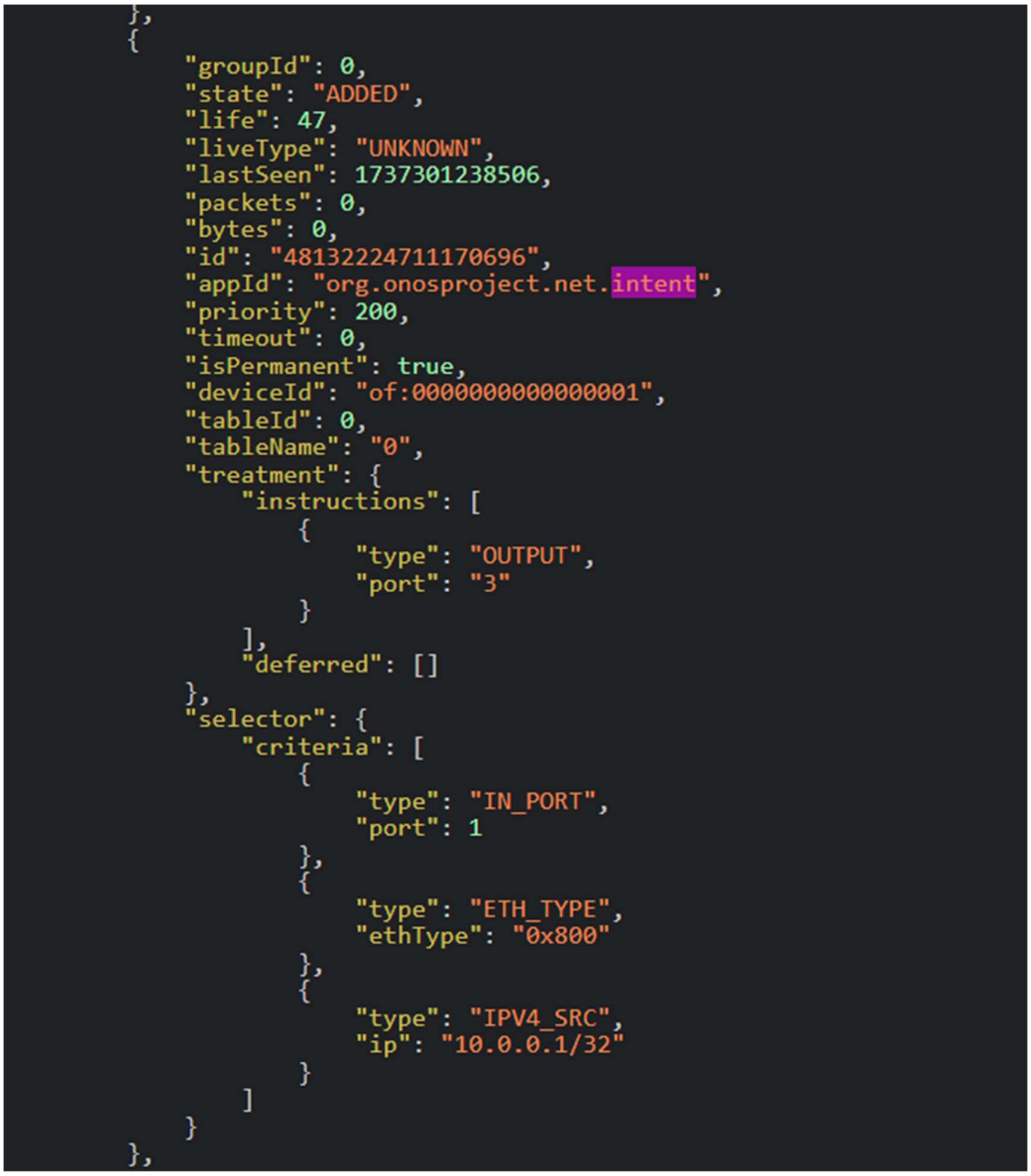


Figure 45 Test Rerouting

The system functionality tests aimed to validate the core features of NetSecFlow, ensuring that traffic control actions such as blocking, allowing, rate-limiting, redirecting, and rerouting traffic were implemented correctly. During the testing, simulated malicious traffic was injected into the network, and the system was instructed to block specific source IP addresses. The block functionality prevented further communication from those sources, confirming the proper execution of traffic control policies. Similarly, the allow function ensured that legitimate traffic was not interrupted. The rate-limiting feature was

tested by applying bandwidth restrictions to certain traffic flows, and the results were verified using network monitoring tools, which confirmed reduced throughput for affected traffic. Redirection to the honeypot was evaluated by isolating suspicious traffic and analyzing its behavior in the controlled environment of the honeypot. Finally, rerouting tests showed that the system could dynamically adjust traffic paths to avoid potential threats while maintaining normal network performance. Each functionality test was successful, proving the system's ability to enforce network policies effectively.

4.5.2 Security Tests

The security tests focused on verifying the robustness of NetSecFlow against various types of cyberattacks and unauthorized access. Some common testing like port scanning, common malware upload was tested with the system. Simulated Distributed Denial of Service (DDoS) attacks were conducted to evaluate the system's ability to maintain operational stability. The rate-limiting feature effectively throttled the attack traffic, ensuring normal network performance for legitimate users. The honeypot was tested for its ability to capture malicious traffic, providing valuable insights without impacting normal network operations. Overall, the security tests demonstrated that NetSecFlow is highly resilient to common threats and ensures secure management of the network.

4.5.3 Integration Tests

Integration tests were conducted to ensure seamless communication between NetSecFlow’s components, including the backend, SDN controller (ONOS), and SIEM tools (Elasticsearch and Kibana). The backend’s APIs were tested to confirm their ability to execute traffic control actions by sending commands to the ONOS controller. For instance, flow rules for blocking and redirecting traffic were successfully implemented on the SDN switches, and their effects were observed in real-time. The ELK stack was tested for its ability to process logs from Suricata and Wazuh, generating actionable

insights and visualizations in Kibana. The integration between the anomaly detection module and the decision-making system was verified by simulating various threat scenarios, which triggered appropriate responses based on the system's marking logic. These tests demonstrated the seamless operation of all components, ensuring consistent and reliable network security management.

4.5.4 Logging and Reporting

Decision Logs

Timestamp	Action Type	Reason	Source IP	Admin/Automated
2025-01-13 02:50:50	Redirect	Redirected traffic from 10.0.0.1 to 10.0.0.6.	10.0.0.1	Automated
2025-01-13 02:50:46	Redirect	Redirected traffic from 10.0.0.1 to 10.0.0.6.	10.0.0.1	Automated
2025-01-13 02:50:46	Redirect	Redirected traffic from 10.0.0.1 to 10.0.0.6.	10.0.0.1	Automated
2025-01-13 02:50:45	Redirect	Redirected traffic from 10.0.0.1 to intermediate switch of:0000000000000003.	10.0.0.1	Automated
2025-01-13 02:50:45	Rate Limit	Rate limited 10.0.0.1 (0A:47:01:10:4D:6D) to 50 bps.	10.0.0.1	Automated
2025-01-13 02:50:45	Rate Limit	Rate limited 10.0.0.1 (0A:47:01:10:4D:6D) to 50 bps.	10.0.0.1	Automated
2025-01-13 02:46:59	Block	Blocked traffic for 10.0.0.1/32 to 10.0.0.2/32.	10.0.0.1/32	Admin
2025-01-13 02:36:11	Block	Blocked traffic for 10.0.0.1/32 to None.	10.0.0.1/32	Automated
2025-01-13 02:11:22	Block	Blocked traffic for 10.0.0.1/32 to None.	10.0.0.1/32	Automated
2025-01-13 02:09:12	Redirect	Redirected traffic from 10.0.0.1 to 10.0.0.6.	10.0.0.1	Automated
2025-01-13 02:09:11	Redirect	Redirected traffic from 10.0.0.1 to intermediate switch of:0000000000000003.	10.0.0.1	Automated

Recent Anomalies

Timestamp	Source IP	Anomaly Type	Score	Action
2025-01-13 01:24:13	10.0.0.2	Port Scanning	31	Investigate
2025-01-03 10:00	10.0.0.11	Port Scanning	75	Investigate
2024-10-01 10:00	10.0.0.11	Port Scanning	75	Investigate
2024-09-01 11:00	192.168.1.25	Unusual Process	60	Investigate
2024-09-01 10:45	192.168.1.15	DNS Spike	80	Investigate
2024-09-01 10:30	192.168.1.30	Malware Upload	85	Investigate
2024-09-01 10:15	192.168.1.20	Brute Force Attempt	90	Investigate

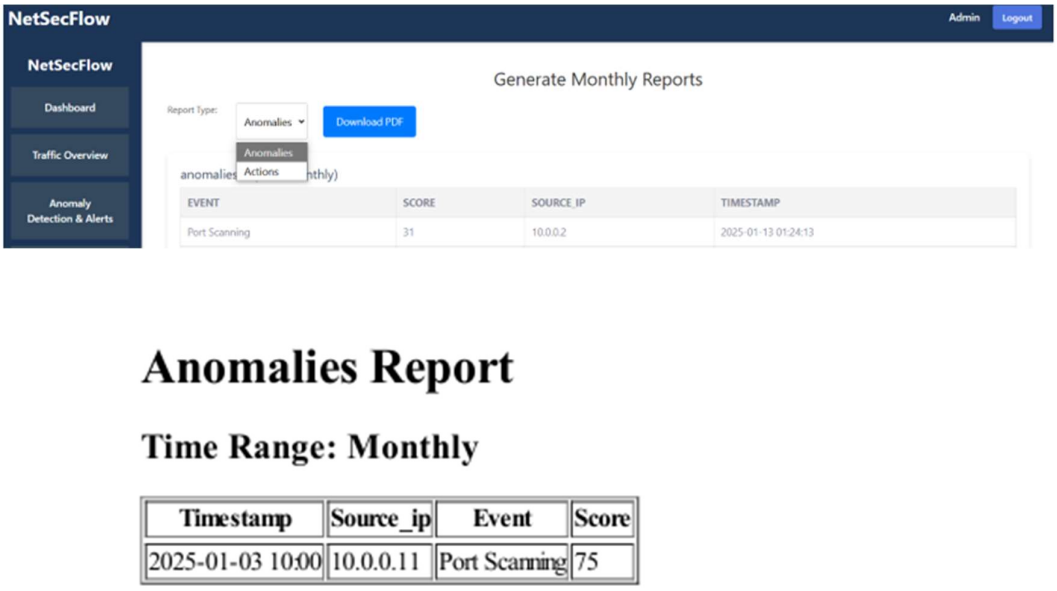


Figure 46 Screenshots of Reports and Logging on the system

Logging and reporting mechanisms were evaluated for their ability to provide detailed and actionable information. The system logs all traffic control actions, including blocking, rate-limiting, and redirection, along with the reasons for these actions. Anomaly detection events are recorded with timestamps, source and destination details, and severity scores, ensuring transparency and traceability. Reports generated included summaries of anomalies detected and actions taken.

4.5.5 Overall System Validation

The overall system validation encompassed end-to-end testing of NetSecFlow in a controlled environment to simulate real-world scenarios. By combining functionality, performance, security, and integration tests, the evaluation confirmed that the system met its design objectives. NetSecFlow effectively identified and mitigated threats, maintained network performance, and provided administrators with intuitive tools for managing security. The system's logging and reporting features enhanced its transparency and accountability, while its scalability and resilience ensured its suitability for modern SDN environments. The validation process highlighted NetSecFlow’s strengths and identified

areas for future improvements, such as enhancing detection algorithms and optimizing resource usage in the ELK stack. These findings demonstrate that NetSecFlow is a robust, efficient, and reliable solution for automated network security management in SDN environments.

4.6 Discussion

4.6.1 User Experience

The **user experience** of the NetSecFlow system is designed with simplicity, efficiency, and effectiveness in mind. The user interface is intuitive and provides administrators with all the necessary tools to monitor and control the network in real time. The dashboard serves as the central hub for managing the system, offering a clear and concise overview of network health, traffic statistics, and anomalies. Visual elements such as charts and graphs are used extensively to make complex data easily understandable, even for users without advanced technical expertise.

The **Traffic Control Page** offers a streamlined process for managing network actions, including blocking, redirecting, rate-limiting, and allowing traffic. Administrators can execute these actions with just a few clicks, and the interface ensures that all options are clearly labeled and accessible. The form dynamically adjusts based on the selected action, displaying only the relevant fields, which eliminates unnecessary complexity and helps users focus on the task at hand.

An essential feature of the system is its **Anomaly Detection Page**, where users can quickly identify and respond to potential threats. Alerts are presented in an organized format, with details about the nature of the anomaly, its source, and suggested actions. This page empowers administrators to act decisively, supported by actionable insights generated by the system's detection mechanisms.

The **Logs and Reports** section enhances user experience by providing detailed records of all actions taken and system events. These logs can be easily accessed and filtered, making it convenient for users to review past decisions, audit system performance, and ensure compliance with organizational policies. The ability to generate detailed reports directly from the system further simplifies the administrative workload.

Finally, the **Chatbot Assistant** adds a layer of accessibility and guidance, especially for new users or those unfamiliar with certain features. The chatbot offers step-by-step instructions, answers common questions, and provides recommendations, ensuring that users can quickly learn to navigate the system and make the most of its capabilities. Together, these features create a cohesive and user-friendly experience that simplifies network security management while maximizing the system's effectiveness.

4.6.2 System Complexity

The **Detection and Decision-Making Processes** in NetSecFlow involve both passive and active methods to identify potential network threats. **Passive Detection** relies on analyzing telemetry data, DNS logs, honeypot logs, and system logs to uncover anomalies. This process leverages the power of ELK (Elasticsearch, Logstash, Kibana) for pattern analysis, ensuring unusual behaviors are flagged while minimizing false positives. By continuously monitoring network activity in a non-intrusive manner, passive detection forms the foundation of early threat identification.

On the other hand, **Active Detection** employs real-time monitoring tools like Suricata for deep packet inspection and Wazuh for endpoint detection. These systems correlate multiple Indicators of Compromise (IoC) to validate threats before generating alerts. The active detection mechanism is highly dynamic and finely tuned to provide actionable intelligence, reducing the likelihood of false alarms while ensuring a rapid response to genuine threats.

NetSecFlow also excels in **Flow Routing and Policy Configuration**, allowing the system to implement advanced mitigation strategies. It supports traffic rerouting to honeypots for isolation, rate limiting to control bandwidth, mirroring for detailed traffic inspection, and eventually blocking malicious traffic. These flexible routing options

ensure that threats can be neutralized without impacting legitimate traffic, maintaining overall network performance and security.

The system's **Decision Module** uses a scoring and marking system that aggregates inputs from different detection methods. Instead of reacting to a single anomaly, such as a DNS traffic spike, the system flags and continuously monitors the suspicious traffic. Only when multiple indicators align does it escalate actions such as blocking or isolation. This measured approach ensures that security measures are robust, adaptive, and precise, avoiding unnecessary disruptions in the network.

4.6.3 Error Free and Error Handling

```
try:
    rate_limit(device_id, rate_limit_bps, src_ip, dst_ip, in_port, out_port)
    return jsonify({"message": f"Rate limit applied from {src_ip} to {dst_ip} at {rate_limit_bps} bps"}), 200
except Exception as e:
    return jsonify({"error": str(e)}), 500
```

Figure 47 Example for error handling

The system incorporates robust error handling and logging mechanisms to ensure reliability and accountability. The **try-except blocks** in the backend code handle scenarios where disallowed or invalid API calls are made. By catching these exceptions, the system prevents unexpected crashes and provides meaningful error messages to users and developers. For example, if an API call attempts to interact with a resource that is unavailable or restricted, the system logs the error and returns a response indicating the issue, ensuring smooth operation without service interruptions.

To enhance security and maintain data integrity, the system employs **input validation** for all API requests. Input validation checks ensure that all parameters received by the backend adhere to expected formats, types, and constraints. This helps to prevent common vulnerabilities such as injection attacks or malformed requests. By rejecting invalid inputs

at the entry point, the system safeguards its internal components and maintains the integrity of operations.

```python
# Helper: Validate IP address format
def is_valid_ip(ip):
    try:
        parts = ip.split('.')
        return len(parts) == 4 and all(0 <= int(part) <= 255 for part in parts)
    except ValueError:
        return False


# Helper: Validate port numbers
def is_valid_port(port):
    return 1 <= port <= 65535
```

Figure 48 Example of Error Handling

The system also features a comprehensive **logging mechanism** to track all actions and errors. Every action performed—whether initiated by a user or triggered automatically—is recorded in the logs, providing a complete audit trail for administrative purposes. Errors, including those encountered during API interactions or other operations, are logged with detailed context to facilitate debugging and future improvements. This logged data not only aids in troubleshooting but also serves as a valuable resource for compliance and performance monitoring.

In the deployment architecture, the system leverages a **containerized cluster** for components like ONOS (Open Network Operating System) and Atomix. This containerized environment ensures scalability, resilience, and efficient resource utilization. ONOS, as the SDN controller, handles network flow management, while Atomix provides distributed coordination and state management for high availability. By running these components in a cluster, the system achieves fault tolerance and can handle dynamic scaling based on demand. This design ensures that the system remains responsive and reliable, even in large or complex network environments.

4.7 Summary

This chapter presents the implementation and evaluation of NetSecFlow. The implementation is described in detail by displaying the user interface and explaining the flow of NetSecFlow when it receives a webhook from SIEM. Evaluation has been done on NetSecFlow. The test results proved that configurations done automatically by NetSecFlow are correct and there are no errors that occurred in the process.

CHAPTER 5: CONCLUSION AND FUTURE WORK

5.1 Introduction

In Chapter 4, the proposed NetSecFlow application and evaluation is presented. This chapter will present the conclusion and discuss future works.

5.2 Conclusion

To summarize, the suggested architecture effectively illustrates how sophisticated security management may be incorporated into a Software Defined Network (SDN) to improve automatic reaction mechanisms to possible security threats. This arrangement provides timely, educated reactions to security anomalies by employing a dynamic and intelligent system consisting of decision-making, configuration creation, and data recording modules, therefore reducing risk and improving network integrity. The system's ability to connect smoothly with existing SIEM/SOAR tools and use telemetry data strengthens its position as a reliable answer to modern network security concerns.

5.3 Future Works

In the future, numerous modifications might be investigated to improve the system's efficacy and application. To begin, adopting more powerful machine learning algorithms may give deeper insights into attack patterns and predictive analytics, enabling more proactive security measures and allowing AI to make actions on its own. Second, improving the system's interoperability to include a broader range of SIEM applications and SDN controller will provide greater applicability across a variety of network infrastructures. Furthermore, investigating decentralized control systems may improve the system's resilience and scalability, especially in bigger, more complicated network contexts. Finally, continuing refining of data recording and analysis procedures would improve forensic capabilities, making it simpler to trace back security incidents and understand their core causes in depth.

REFERENCES

Documentation (no date) Elasticsearch Platform - Find real-time answers at scale. Available at: https://www.elastic.co/docs (Accessed: 14 June 2024).

Documentation (no date) Documentation - Splunk Documentation. Available at: https://docs.splunk.com/Documentation (Accessed: 14 June 2024).

Foundation, O.N. (no date) Open network operating system (ONOS), onosproject. Available at: https://docs.onosproject.org/ (Accessed: 14 June 2024).

Getting started¶ (no date) Getting Started - Open vSwitch 3.3.90 documentation. Available at: https://docs.openvswitch.org/en/latest/intro/ (Accessed: 14 June 2024).

Hammad, Mohamed & Hewahi, Nabil & Elmedany, Wael. (2023). Enhancing Network Intrusion Recovery in SDN with machine learning: an innovative approach. Arab Journal of Basic and Applied Sciences. 30. 561-572. 10.1080/25765299.2023.2261219.

Home (2024) Docker Documentation. Available at: https://docs.docker.com/ (Accessed: 14 June 2024).

Jiménez, M. B., Fernández, D., Rivadeneira, J. E., Bellido, L. L., & Cárdenas, A. F. P. (2021). A survey of the main security issues and solutions for the SDN architecture. IEEE Access, 9, 122016–122038. https://doi.org/10.1109/access.2021.3109564

M. Ehrlich, L. Wisniewski, H. Trsek, D. Mahrenholz and J. Jasperneite, "Automatic mapping of cyber security requirements to support network slicing in software-defined networks," 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Limassol, Cyprus, 2017, pp. 1-4, doi: 10.1109/ETFA.2017.8247728.

keywords: {Computer security;Quality of service;Standards;Automation;Industries},

Python 3.12.4 documentation (no date) 3.12.4 Documentation. Available at: https://docs.python.org/3/ (Accessed: 14 June 2024).

Software-defined networking (SDN) definition (2023) Cisco. Available at: https://www.cisco.com/c/en/us/solutions/software-defined-networking/overview.html#~stickynav=3 (Accessed: 11 May 2024).

SDN and orchestration: Juniper Networks Us (no date) Juniper Networks. Available at: https://www.juniper.net/us/en/products/sdn-and-orchestration.html (Accessed: 11 May 2024).

Suricata User guide (no date) Suricata User Guide - Suricata 8.0.0-dev documentation. Available at: https://docs.suricata.io/en/latest/ (Accessed: 14 June 2024).

Snort setup guides for emerging threats prevention. Available at: https://www.snort.org/documents (Accessed: 14 June 2024).

Ubuntu documentation. Available at: https://docs.ubuntu.com/ (Accessed: 14 June 2024).

Wazuh (no date) WAZUH documentation, Wazuh documentation. Available at: https://documentation.wazuh.com/current/index.html (Accessed: 14 June 2024).

APPENDICES

Appendix 1(Source Code):

https://github.com/heheJY/netsecflow/

Appendix 2(Example Intent File generated during the decision process):


```
{
   "type": "PointToPointIntent",
   "appId": "org.onosproject.cli",
   "priority": 200,
   "selector": {
      "criteria": [
         {
            "type": "ETH_TYPE",
            "ethType": "0x0800"
         },
         {
            "type": "IPV4_SRC",
            "ip": "10.0.0.1/32"
         }
```

```
        ]
    },
    "ingressPoint": {
        "port": "1",
        "device": "of:0000000000000001"
    },
    "egressPoint": {
        "port": "3",
        "device": "of:0000000000000001"
    }
}
```