People's Democratic Republic of Algeria
Ministry of higher education and scientific research
University of Kasdi Merbah Ouargla

Faculty of New Technologies of Information and Communication
Department of Computer Science and Information Technology

**Master Thesis**

**Domain:** Mathematics and Computer Science

**Sector:** Computer Science

**Specialty:** Network Administration and Security

***Theme***

# Adaptive Lightweight Flow-Based Detection and Mitigation of DDoS and Scanning Attacks in Software-Defined Networks

**Presented by:**

Mazri Khadidja

**Publicly discussed:** 08/06/2026

**Jury Members:**

| | | |
|---|---|---|
| **Dr.** Mohammed Kamel Benkaddour | President | University of Ouargla |
| **Dr.** Ahmida Djediai | Examiner | University of Ouargla |
| **Dr.** Fares Kahlessenane | Supervisor | University of Ouargla |

**Academic year: 2025/2026**

# Acknowledgments

First and foremost, I would like to thank Allah Almighty for giving me the strength, patience, health, and clarity of mind to complete this thesis, and by His grace alone I was able to bring this work to completion.

I express my deepest gratitude to my supervisor **Dr. Fares Kahlessenane** for his continuous guidance, valuable advice, and encouragement throughout this work.

I would also like to thank the members of the jury for accepting to examine and evaluate this work.

# Dedication

**To my beloved mother,** my first teacher and the most wonderful human being I have ever known. You are the source of my principles, my endless comfort, and the ultimate inspiration behind everything I achieve. Your quiet sacrifices and constant prayers have been my strength and comfort through every long night of this Master's journey. As I step forward into my next adventure, I carry your values with me, bound by a promise to make you proud. I love you, mommy.

***To my dear father,*** words cannot fully express my gratitude for a strong, exceptional, and hardworking man who has always been my protector and my truest friend. Thank you for your endless encouragement and for always believing in my potential, even when things got difficult. Your unwavering faith in me gave me the strength to stand tall and conquer every challenge in my academic path. As I step into the future, I promise to carry your strength with me and make you proud. I dedicate this milestone to your love and lifelong support. I love you, baba

**To my family and friends,** thank you for being my pillars of strength and the joy that kept me moving forward. To my sisters and lovely little brothers, your smiles and support turned every challenge into a milestone. This achievement belongs to you all.

# ملخص

ملخص تُمركز الشبكات المعرّفة بالبرمجيات ذكاء الشبكة داخل وحدة تحكم واحدة، مما يوفر قابلية البرمجة والرؤية الشاملة للشبكة. غير أن هذه البنية تُفرز ثغرة هيكلية حرجة: يصبح مستوى التحكم موردًا محدودًا قابلًا للاستغلال، يؤدي استنزافه عبر إغراق رسائل Packet-In إلى تعطّل الشبكة بأكملها. تستغل هجمات الحرمان من الخدمة الموزّعة هذه التبعية بتوليد حركة مرور حجمية ضخمة نحو هدف واحد، بينما تقوم هجمات المسح الشبكي بفحص عدد كبير من الوجهات المتمايزة لاستدعاء أحداث table-miss متواصلة، مما يُرغم وحدة التحكم على استنزاف مواردها بحيث لا يمكن معالجة أي تدفق شرعي. تقترح هذه الأطروحة نظامًا خفيفًا وتكيفيًا للكشف والتخفيف يعتمد على تحليل التدفقات، مدمجًا مباشرةً في وحدة التحكم OS-Ken، وتُنفّذه وتُقيّمه. يستخلص النظام ستة مقاييس سلوكية لكل مصدر من إحصاءات تدفق OpenFlow الأصلية، ويُطبّق عتبات تكيّفية مبنية على المتوسط المتحرك ذي الترجيح الأسي، تتتبع باستمرار السلوك الفردي لكل مصدر دون أي مرحلة تدريب أو مجموعة بيانات مُصنّفة. تُميّز شروط تصنيف محددة رسميًا بين هجمات DDoS وهجمات المسح بناءً على أنماط مرور إحصائية متعامدة. عند التأكيد، تُثبّت وحدة التخفيف المستقلة قواعد DROP ذات أولوية عبر خط أنابيب OpenFlow ثنائي الجدول مع استرداد تلقائي قائم على انتهاء المهلة دون تدخل المشغّل. أثبت النظام، في بيئة SDN مضبوطة تعتمد على Mininet، معدل إيجابيات كاذبة يساوي الصفر عبر جميع مصادر حركة المرور الشرعية، ونجح في اكتشاف هجمات DDoS والمسح الشبكي وحجبهما في دورة استطلاع واحدة مدتها ثانية واحدة. ظل العبء الحسابي ضئيلًا طوال التجارب، بمتوسط استخدام للمعالج بلغ 3.73% وبصمة ذاكرة مستقرة عند 60.1 ميغابايت. تُنفَّذ جميع عمليات الكشف في زمن خطي O(F) لكل دورة استطلاع، مما يؤكد أن النهج المقترح يوفر حماية سريعة ومستقلة ضد تهديدات DDoS والمسح دون المساس بأداء وحدة التحكم

**الكلمات المفتاحية:** الشبكات المعرّفة بالبرمجيات، كشف DDoS، المسح الشبكي، EWMA التكيّفي، OpenFlow، كشف التسلل القائم على التدفقات، أمن مستوى التحكم، التخفيف المستقل.

# Abstract

Software-Defined Networking centralizes network intelligence within a single controller, delivering programmability and global network visibility. However, this architecture introduces a critical structural vulnerability: the control plane becomes a finite, exploitable resource whose exhaustion through Packet-In flooding renders the entire network inoperable. Distributed Denial of Service attacks exploit this dependency by generating massive volumetric traffic toward a single target, while network scanning attacks probe large numbers of distinct destinations to trigger continuous table-miss events, forcing the controller into resource exhaustion from which no legitimate flow can be processed. This thesis proposes, implements, and evaluates a lightweight adaptive flow-based detection and mitigation system integrated directly into the OS-Ken SDN controller. The framework derives six per-source behavioral metrics from native OpenFlow flow statistics and applies Exponentially Weighted Moving Average adaptive thresholds that continuously track individualized source behavior without any training phase or labeled dataset. Formally defined classification conditions distinguish DDoS from scanning attacks based on their statistically orthogonal traffic profiles. Upon confirmed detection, an autonomous mitigation module installs priority DROP rules across a two-table OpenFlow pipeline with timeout-based automatic recovery, requiring no operator intervention. Evaluated in a controlled Mininet-based SDN testbed, the system maintained a false positive rate of zero across all legitimate traffic sources and successfully detected and blocked both DDoS flooding and network scanning within a single one-second polling cycle. Computational overhead remained negligible throughout all experiments, with an average CPU utilization of 3.73% and a stable memory footprint of 60.1 MB. All detection operations execute in linear time O(F) per polling cycle, confirming that the proposed approach provides rapid and autonomous protection against DDoS and scanning threats without compromising controller performance.

**Keywords:** Software-Defined Networking, DDoS detection, network scanning, adaptive EWMA, OpenFlow, flow-based intrusion detection, control plane security, autonomous mitigation.

# Résumé

Les réseaux définis par logiciel centralisent l'intelligence réseau au sein d'un contrôleur unique, offrant programmabilité et visibilité globale du réseau. Cependant, cette architecture introduit une vulnérabilité structurelle critique : le plan de contrôle devient une ressource finie et exploitable dont l'épuisement par inondation de messages Packet-In rend l'ensemble du réseau inopérant. Les attaques par déni de service distribué exploitent cette dépendance en générant un trafic volumétrique massif vers une cible unique, tandis que les attaques de balayage réseau sondent un grand nombre de destinations distinctes pour déclencher des événements table-miss continus, contraignant le contrôleur à un épuisement de ses ressources depuis lequel aucun flux légitime ne peut être traité. Cette thèse propose, implémente et évalue un système léger de détection et de mitigation adaptatif basé sur les flux, intégré directement au contrôleur OS-Ken. Le cadre dérive six métriques comportementales par source à partir des statistiques de flux OpenFlow natives et applique des seuils adaptatifs basés sur la moyenne mobile à pondération exponentielle qui suivent en continu le comportement individualisé de chaque source sans phase d'entraînement ni ensemble de données étiqueté. Des conditions de classification formellement définies distinguent les attaques DDoS des attaques de balayage selon leurs profils de trafic statistiquement orthogonaux. Lors d'une détection confirmée, un module de mitigation autonome installe des règles DROP prioritaires dans un pipeline OpenFlow à deux tables avec récupération automatique basée sur l'expiration, sans intervention de l'opérateur. Évalué dans un environnement SDN contrôlé sous Mininet, le système a maintenu un taux de faux positifs nul sur toutes les sources de trafic légitime et a détecté et bloqué avec succès les attaques DDoS et le balayage réseau en un seul cycle d'interrogation d'une seconde. La surcharge computationnelle est restée négligeable tout au long des expériences, avec une utilisation moyenne du CPU de 3,73% et une empreinte mémoire stable de 60,1 Mo. Toutes les opérations de détection s'exécutent en temps linéaire O(F) par cycle d'interrogation, confirmant que l'approche proposée fournit une protection rapide et autonome contre les menaces DDoS et de balayage sans compromettre les performances du contrôleur.

**Mots clés :** Réseaux définis par logiciel, détection DDoS, balayage réseau, EWMA adaptatif, OpenFlow, détection d'intrusion basée sur les flux, sécurité du plan de contrôle, mitigation autonome.

# Table of Contents

**Acknowledgements** I
**Dedication** II
**ملخص** III
**Abstract** IV
**Résumé** V
**Table of Contents** VI
**List of Figures** IX
**List of Tables** X
**List of Acronyms** XI
**General Introduction** 1

**Chapter 1 : Theoretical Background and Foundational Concepts** 5
1.1 Introduction 5
1.2 Software-Defined Networking (SDN) Overview 6
1.2.1. SDN Architecture: Data Plane, Control Plane, Application Plane 6
1.2.1.1. Data Plane 6
1.2.1.2. Control Plane 7
1.2.1.3. Application Plane 9
1.2.2. Southbound and Northbound APIs (OpenFlow, P4) 10
1.2.2.1. Southbound API 10
1.2.2.2. Northbound API 12
1.3 Distributed Denial of Service (DDoS) Attacks in SDN 12
1.3.1. DDoS Attack Mechanisms and Taxonomy 12
1.3.1.1. Definition 12
1.3.1.2. Taxonomy 13
1.3.2. Impact of DDoS Attacks on SDN Architecture 15
1.3.3. Horizontal and Vertical Scanning 17
1.3.4. Flow-Level Behavior and Impact on SDN 18
1.4 Conclusion 19

**Chapter 2 : State of the Art: Intrusion Detection Mechanisms in SDN** 20
2.1 Introduction 20
2.2 Detection Approaches for Network Security 21
2.2.1. Machine Learning and Deep Learning Based Detection 21
2.2.1.1. Classical Machine Learning 21
2.2.1.2. Deep Learning Architectures 22
2.2.1.3. Lightweight Variants 22
2.2.1.4. Limitations in SDN Deployment Contexts 23
2.3 Lightweight Anomaly Detection Methods 24
2.3.1. Entropy-Based Detection 24
2.3.2. Statistical Detection Methods (CUSUM, EWMA) 25
2.3.3. Feature Selection for Lightweight Detection 27
2.3.4. Rule-Based and Threshold Detection 28
2.3.5. Limitations of Lightweight Methods 28

2.4 Related Work on Detection and Mitigation Systems in SDN ……. 29
2.4.1. DDoS Detection and Mitigation Systems …… 29
2.4.2. Scanning Detection Systems in SDN …… 30
2.4.3. Hybrid and Flow-Based Analysis Systems …… 31
2.5 Summary of Prominent Work ……. 32
2.6 Comparative Analysis ……. 33
2.7 Research Gap ……. 34
2.8 Conclusion …… 34

**Chapter 3 : A Lightweight Flow-Based Threat Detection and Mitigation in SDN ……..** 36
3.1 Introduction ……. 36
3.2 Overview of the Proposed System …….. 37
3.3 Flow-Level Feature Extraction …….. 38
3.3.1. OpenFlow Statistics Collection Mechanism …… 38
3.3.2. Feature Definition and Behavioral Significance …… 39
3.4 Adaptive Baseline Estimation Using EWMA …….. 40
3.4.1. Per-Source State Initialization and Warm-Up …… 40
3.4.2. Threshold Derivation and Update Rule …… 41
3.4.3. Justification for EWMA as the Baseline Estimator …… 42
3.5 Attack Classification Logic …….. 43
3.5.1. DDoS Detection Conditions …… 43
3.5.2. Scanning Detection Conditions …… 44
3.5.3. Anomaly Scoring and Alert Emission …… 44
3.6 The Two-Table OpenFlow Mitigation Pipeline …….. 45
3.6.1. Pipeline Architecture and Flow Entry Design …… 45
3.6.2. DROP Rule Installation, Expiry, and Renewal …… 46
3.7 Computational Complexity Analysis …….. 47
3.8 Conclusion …… 48

**Chapter 4 : Experimental Evaluation and Results ……..** 50
4.1 Introduction ……. 50
4.2 Simulation Environment …….. 51
4.2.1. Hardware and Virtualization Platform …… 51
4.2.2. Software Stack …… 51
4.2.3. Network Topology …… 51
4.3 Traffic Generation Methodology …….. 52
4.3.1. Normal Traffic Generation …… 52
4.3.2. DDoS Traffic Generation …… 53
4.3.3. Scanning Traffic Generation …… 53
4.4 Attack Scenarios …….. 53
4.5 Evaluation Metrics …….. 54
4.5.1. Detection Performance Metrics …… 54
4.5.2. Operational Efficiency Metrics …… 54
4.6 Normal Traffic Baseline Results …….. 55
4.7 Scanning Attack Detection and Mitigation (S3) …….. 58
4.7.1. Detection Performance …… 58
4.7.2. Mitigation Effectiveness …… 59
4.8 DDoS Attack Detection and Mitigation (S1/S2) …….. 61
4.8.1. Detection Performance…… 61

4.8.2. Mitigation Effectiveness ……..……...…… 64
4.9 Overall Detection Performance Summary ……..……..…..……. 66
4.10 Operational Efficiency ……..……..……. 67
4.10.1. CPU Utilization ……..…… 67
4.10.2. Memory Footprint ……..…… 67
4.10.3. Controller Throughput ……...…… 68
4.11 Research Gap Validation ……..……...……. 68
4.12 Discussion ……..……. 69
4.13 Conclusion ….…… 70

**General conclusion ……..** 71

# List of figures


Fig 1.1: SDN Three-Plane Architecture …… 6
Fig 1.2: Internal Architecture of the SDN Control Plane and its Interactions with the Application and Data Planes ……..…….…. 8
Fig 1.3: DDoS attack scenario in an SDN network. Botnet traffic is forwarded through the OpenFlow switch toward the target, resulting in service unavailability …… 13
Fig 1.4: Global DDoS Attack Trends (2020–2023) ……...…… 15
Fig 1.5: Resource exhaustion during a DDoS attack. (a) Displays the rapid saturation of switch TCAM; (b) Illustrates the exponential increase in controller CPU utilization ……...…… 16
Fig 1.6: Scanning Attack Scenario in SDN: From Probe Generation to Controller Notification … 17
Fig 1.7: Global Network Scanning Trends (2020–2024) …… 18
Fig 1.8: SDN Controller Resource Exhaustion Under Scanning Attack Stress ……...…… 19
Fig 2.1: Taxonomy of ML/DL Detection Approaches and Their Computational Overhead ……. 21
Fig 2.2: Shannon entropy H(X) under normal and attack traffic conditions ……... 25
Fig 3.1: High-level overview of the proposed adaptive detection system ……... 38
Fig 3.2: Two-table OpenFlow pipeline showing packet processing sequence and Flow-Mod installation paths ……. 46
Fig 4.1: Tree (2,3) evaluation topology with host role assignments …… 52
Fig 4.2: SDN controller status and mitigation panel during normal traffic operation ……. 56
Fig 4.3: Anomaly score and destination entropy evolution during normal traffic baseline …….. 56
Fig 4.4: Controller terminal log showing system initialization and normal traffic classification ….. 57
Fig 4.5: Temporal evolution of anomaly score and destination entropy during scanning scenario S3 ……... 58
Fig 4.6: Flow count, packet rate, and destination diversity metrics during scanning scenario S3 … 59
Fig 4.7: Controller mitigation status and SDN classification output during scanning attack detection ……. 60
Fig 4.8: OS-Ken controller log output illustrating the complete scanning detection and automated mitigation lifecycle ……. 61
Fig 4.9: Anomaly score and entropy behavior during DDoS scenario S1/S2 ……. 62
Fig 4.10: Flow count and packet rate dynamics during DDoS scenario S1/S2 ……... 63
Fig 4.11: Destination count scan indicator during DDoS scenario confirming single-target flooding ……... 63
Fig 4.12: Controller mitigation status and SDN classification output during DDoS attack detection ……. 64
Fig 4.13: Post-attack system recovery confirming automated source unblocking and status restoration ……... 65
Fig 4.14: OS-Ken controller log output illustrating the complete DDoS detection and automated mitigation lifecycle ……... 66
Fig 4.15: Confusion matrices for the three evaluation scenarios: normal traffic baseline, scanning (S3), and DDoS (S1/S2) …….. 66
Fig 4.16: OS-Ken controller process CPU utilization over a 60-second monitoring interval …….. 67

# List of tables


Tab 1.1: OpenFlow vs P4: Southbound API Comparison ……. 11
Tab 1.2: DDoS Attack Taxonomy by Protocol Layer and Exhaustion Strategy ……...……... 14
Tab 2.1: Computational Complexity of ML/DL Approaches per Detection Cycle ……..... 23
Tab 2.2: Flow-level features for lightweight anomaly detection in SDN ……..... 27
Tab 2.3: Most Prominent Works in SDN Intrusion Detection ……...….... 32
Tab 2.4: Comparative Analysis of SDN Intrusion Detection and Mitigation Systems ……..... 33
Tab 3.1: Feature Behavioral Divergence Across Traffic Classes ……. 40
Tab 3.2: Comparison of Baseline Estimation Methods for SDN Anomaly Detection ……. 43
Tab 3.3: DDoS Classification Conditions ……. 43
Tab 3.4: Scanning Classification Conditions ……. 44
Tab 3.5: Computational Complexity Comparison per Detection Cycle ……. 48
Tab 4.1: Experimental Software Stack ……... 51
Tab 4.2: Attack Scenarios ……..... 53
Tab 4.3: Operational Efficiency Metrics ……..... 55
Tab 4.4: Normal Traffic Baseline Classification Results …….. 57
Tab 4.5: Operational Efficiency Results …… 68

# List of Acronyms

| | |
|---|---|
| **API** | Application Programming Interface |
| **ASIC** | Application-Specific Integrated Ciruit |
| **CNN** | Convolutional Neural Network |
| **CUSUM** | Cumulative Sum Control Chart |
| **DDoS** | Distributed Denial of Service |
| **DL** | Deep Learning |
| **DoS** | Denial of Service |
| **DT** | Decision Tree |
| **EWMA** | Exponentially Weighted Moving Average |
| **FN** | False Negative |
| **FP** | False Positive |
| **FPGA** | Field-Programmable Gate Array |
| **FSM** | Finite State Machine |
| **GRU** | Gated Recurrent Unit |
| **IDS** | Intrusion Detection System |
| **INT** | In-band Network Telemetry |
| **IPS** | Intrusion Prevention System |
| **LLDP** | Link Layer Discovery Protocol |
| **LSTM** | Long Short-Term Memory |
| **ML** | Machine Learning |
| **NFV** | Network Functions Virtualization |
| **NIDS** | Network Intrusion Detection System |
| **NOS** | Network Operating System |
| **ONF** | Open Networking Foundation |
| **ONOS** | Open Network Operating System |
| **OVS** | Open vSwitch |
| **P4** | Programming Protocol-Independent Packet Processors |
| **QoS** | Quality of Service |

| | |
|---|---|
| **REST** | Representational State Transfer |
| **RF** | Random Forest |
| **RSS** | Resident Set Size |
| **RTT** | Round-Trip Time |
| **SDN** | Software-Defined Networking |
| **SPOF** | Single Point of Failure |
| **SPRT** | Sequential Probability Ratio Test |
| **SVM** | Support Vector Machine |
| **TCAM** | Ternary Content Addressable Memory |
| **TN** | True Negative |
| **TP** | True Positive |
| **TRW** | Threshold Random Walk |

# General Introduction

## Background and Context

Modern computer networks have undergone significant evolution over the past decades, driven by the rapid expansion of cloud computing, mobile communications, and data-intensive applications. As network infrastructures continue to scale, they are required to support increasingly dynamic and heterogeneous environments. Traditional network architectures, however, were not originally designed to accommodate such levels of flexibility and scalability. They rely on distributed control mechanisms, where each network device independently makes forwarding decisions based on locally stored configurations, resulting in complex, rigid, and labor-intensive network management [1].

In conventional networking paradigms, the control plane and the data plane are tightly coupled within the same proprietary devices. While this design has historically ensured stability and interoperability, it introduces critical limitations in current network environments. Network configuration requires manual intervention across multiple devices, the absence of a global network view restricts traffic optimization, and the lack of centralized programmability slows down the deployment of new services. These challenges have driven the need for a more flexible and manageable networking paradigm capable of meeting the demands of modern digital infrastructures.

To address these limitations, Software-Defined Networking has emerged as a transformative approach that decouples the control plane from the data plane, centralizing network intelligence in a dedicated software controller. This architectural shift enables network administrators to manage network behavior through high-level programming interfaces rather than low-level device-specific configurations, providing a global network view and enabling real-time adaptation to evolving traffic conditions [2]. However, the very centralization that makes SDN powerful simultaneously introduces a critical security dependency: the controller becomes a single point of

failure whose exhaustion paralyzes the entire network it manages. Distributed Denial of Service attacks exploit this dependency by flooding the control plane with excessive Packet-In messages, forcing the controller into resource exhaustion from which no legitimate flow can be processed. Network scanning attacks similarly stress the controller by generating large numbers of distinct flows, each triggering a separate controller intervention. In both cases, the attacker does not need to compromise the controller directly it simply needs to exhaust it.

Flow-based network monitoring has emerged as an efficient and suitable foundation for detecting such threats within SDN environments. Instead of inspecting individual packets, this approach relies on aggregated flow-level statistics packet rates, flow counts, destination diversity, and traffic entropy that can be computed directly from native OpenFlow counters already maintained by the controller. By focusing on lightweight aggregated features, flow-based analysis can identify anomalous traffic patterns with minimal processing overhead while leveraging the global visibility already provided by the SDN control plane, making it a natural fit for efficient and scalable security mechanisms in SDN deployments.

## Problem Statement

The centralization of control in Software-Defined Networking (SDN) introduces a critical dependency on the controller as the core decision-making entity. While this design improves network management, it also makes the control plane highly vulnerable to abnormal traffic patterns.

Distributed Denial of Service (DDoS) attacks can overwhelm the controller by generating excessive traffic and flow requests, leading to control plane saturation and service degradation. Similarly, network scanning attacks produce a large number of distinct flows, increasing the controller's processing load while remaining more difficult to distinguish from legitimate behavior.

Existing solutions often rely on machine learning and deep learning techniques that require significant computational resources. This creates a fundamental paradox: the mechanisms intended to protect the controller may themselves contribute to resource consumption and performance degradation.

## Motivation

The increasing reliance on SDN controllers for network management requires security mechanisms that can operate efficiently without degrading performance. Given the susceptibility of the control plane to resource exhaustion, it is essential to develop detection approaches that minimize computational overhead while maintaining real-time responsiveness, and apply real-time mitigation through OpenFlow flow table manipulation. Lightweight flow-based techniques

offer a promising direction, as they rely on simple and scalable features that can be processed efficiently. Additionally, leveraging data plane capabilities for partial analysis or traffic blocking can further reduce the burden on the controller. These considerations motivate the need for efficient, adaptive detection and automated mitigation mechanisms tailored to the constraints of SDN environments.

## Research Objectives and Questions

The objective of this work is to design a lightweight, flow-based detection and mitigation mechanism for identifying and blocking DDoS and scanning attacks in SDN while minimizing controller overhead and ensuring real-time performance.

The study addresses the following research questions:

> **RQ1:** Which flow-level features are most effective for distinguishing DDoS and scanning behaviors in SDN environments?
>
> **RQ2:** How can these features be computed efficiently to minimize overhead on the SDN controller?
>
> **RQ3:** What trade-off exists between detection accuracy, real-time performance, and mitigation effectiveness in terms of blocking speed and traffic reduction in lightweight flow-based detection?

## Research Contributions

In line with the stated objectives, this study proposes a lightweight, flow-based detection approach for Software-Defined Networking (SDN). The method relies on simple flow-level features, such as packet rate, number of flows, and destination diversity, to identify abnormal traffic patterns while maintaining low computational overhead and enabling real-time analysis within the SDN controller.

The main contribution of this thesis is the design of an adaptive detection and mitigation mechanism capable of distinguishing between Distributed Denial of Service (DDoS) and scanning attacks based on their traffic characteristics, and automatically blocking confirmed attackers through an OpenFlow two-table pipeline. Unlike traditional approaches that rely on computationally intensive machine learning models, the proposed solution emphasizes algorithmic efficiency and suitability for resource-constrained control planes.

To address the trade-off between detection accuracy and performance, the following contributions are proposed:

> **Lightweight Flow-Based Detection:** A real-time detection mechanism using aggregated flow statistics with minimal processing overhead.

**Adaptive Thresholding Strategy:** A classification approach that differentiates DDoS and scanning behaviors based on traffic intensity and destination patterns.

**Controller-Efficient Design:** An implementation optimized to preserve SDN controller performance and reduce latency.

**OpenFlow Table-Based Mitigation:** An automated mitigation mechanism using a two-table OpenFlow pipeline that installs DROP rules in a dedicated mitigation table upon attack detection, with configurable timeout-based expiry to ensure automatic unblocking once the threat subsides.

## Roadmap of the Thesis

This thesis is organized as follows:

**Chapter 1** introduces the fundamental concepts of Software-Defined Networking (SDN), including its architecture, planes, and communication interfaces, along with background on DDoS and scanning attacks.

**Chapter 2** reviews the state of the art in SDN security, focusing on detection methods for DDoS and scanning attacks, and highlights the limitations of existing approaches.

**Chapter 3** presents the proposed lightweight flow-based detection and mitigation approach, including the system architecture, feature selection, detection logic, and the OpenFlow two-table mitigation pipeline.

**Chapter 4** presents the experimental setup and results of the proposed lightweight detection and mitigation system. It describes the simulation environment, network topology, and controlled traffic generation, and evaluates detection performance, mitigation effectiveness in terms of blocking speed and traffic reduction, and computational efficiency.

Chapter

# 1 Theoretical Background and Foundational Concepts

## 1.1. Introduction

Software-Defined Networking (SDN) provides a programmable framework for managing network behavior through centralized control and dynamic flow management. This paradigm enables fine-grained visibility of network traffic and supports real-time interaction between the controller and forwarding devices. In such environments, traffic behavior is defined in terms of flows, where each flow represents a set of packets sharing common characteristics. The analysis of these flows plays a central role in understanding network activity and identifying abnormal patterns that may affect system operation. Among these patterns, Distributed Denial of Service (DDoS) attacks and network scanning activities exhibit distinct flow characteristics that directly influence how the controller processes traffic and allocates resources. Accurately capturing and interpreting these behaviors is essential for designing effective and lightweight detection mechanisms in SDN environments. This chapter presents the theoretical background necessary to support the proposed work. It introduces SDN architecture and communication mechanisms, analyzes DDoS and scanning attack behaviors and their specific impact on the SDN control plane, establishing the conceptual foundations on which the proposed detection approach is built.

## 1.2. Software-Defined Networking (SDN) Overview

Software-Defined Networking (SDN) uses a layered architecture that enables programmable control of network behavior through interactions between its components [3]. In this model, forwarding devices manage packets according to flow rules, while control decisions are made externally using a global view of the network. This operation relies on continuous communication between switches and the controller, allowing rules to be dynamically installed and updated in response to traffic conditions. Network policies are translated into control actions, linking decision-making with packet forwarding [4]. This architecture supports flexible traffic management and provides the basis for monitoring and security mechanisms in SDN environments.

### 1.2.1. SDN Architecture: Data Plane, Control Plane, Application Plane

The SDN architecture is organized into three main functional planes that work together to enable programmable network behavior. Each plane plays a distinct role, and their interaction defines how traffic is processed and controlled within the network, as illustrated in Figure 1.1.

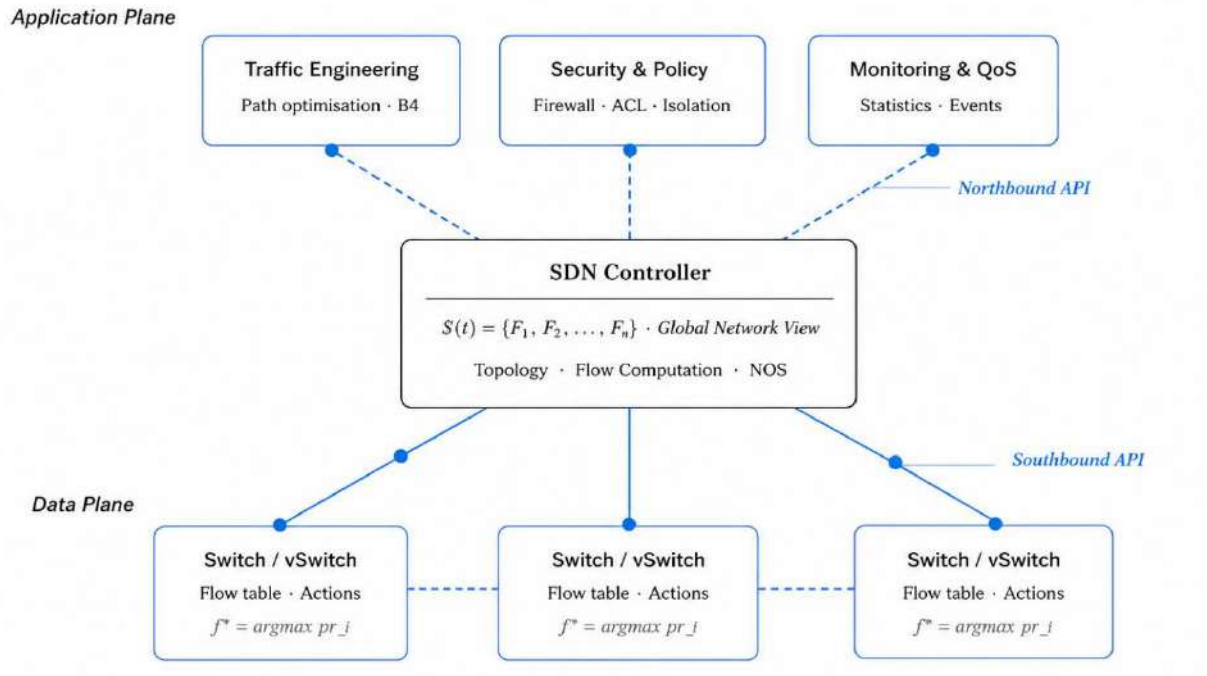


Figure 1.1 – SDN Three-Plane Architecture.

#### 1.2.1.1. Data Plane

The data plane, also referred to as the forwarding plane, is the lowest layer of the SDN architecture. It comprises the physical and virtual network devices switches and routers whose sole responsibility is to forward, drop, or modify packets according to rules installed by the control plane [4][5]. In traditional networks, the forwarding logic and the control logic are tightly

coupled within the same proprietary device. SDN fundamentally breaks this coupling, reducing the data plane to a simple, fast, and programmable forwarding fabric [6].

The core concept of the data plane is the match-action paradigm [7]. Each forwarding device maintains one or more flow tables. Upon packet arrival, the device pulls out the header fields and compares them to the flow table's entries. The entry with the highest priority that matches is chosen, and the corresponding action is carried out. This action could involve forwarding the packet to a specific output port, discarding it, altering a header field, or sending it to the controller. Formally, each flow entry fi is defined as:

$$f_i = \langle m_i, pr_i, A_i, c_i, t_i \rangle \tag{1.1}$$

where $m_i$ is the match pattern over header fields, $pr_i$ is the priority $A_i$ is the action set, $c_i$ are the packet and byte counters, and $t_i$ encodes the timeout constraints [8]. The forwarding decision for an incoming packet with header H is then:

$$f^* = \arg\max pr_i \quad f_i \in \mathcal{F} \;\; m_i \subseteq H \tag{1.2}$$

If no entry matches a table-miss the packet is either dropped or sent to the controller via a Packet-In message for reactive rule installation [8][9]. This abstraction generalizes all forwarding behaviors Ethernet switching, IP routing, VLAN tagging, MPLS into a single unified model, making the data plane both vendor-agnostic and fully programmable [7][10].

#### 1.2.1.2. Control Plane

The control plane is the middle layer of the SDN architecture and represents the intelligence of the network. It is embodied by the SDN controller a software entity that maintains a global view of the network and translates high-level policies from the application plane into low-level forwarding rules installed in the data plane [5][11]. Gude et al. [12] described this role precisely by analogy: the SDN controller functions as a network operating system (NOS), providing a programmatic abstraction of the underlying hardware to applications in the same way an OS abstracts hardware resources to software processes.

A key architectural property of the control plane is logical centralization [5][13]. Although the controller may be physically distributed across multiple machines for scalability and fault tolerance, it presents a single, consistent network view to applications. This logical centralization greatly simplifies network management tasks such as routing, access control, and load balancing can be expressed as global computations over a unified topology graph, rather than as distributed protocols running independently on each device [6][12].

The controller communicates downward with data plane devices through the southbound API most commonly OpenFlow and upward with network applications through the northbound API, typically exposed as REST interfaces or intent-based abstractions [13][14]. In distributed deployments, multiple controller instances synchronize state through east-westbound APIs [12][15]. This interface structure is illustrated in Figure 1.2.

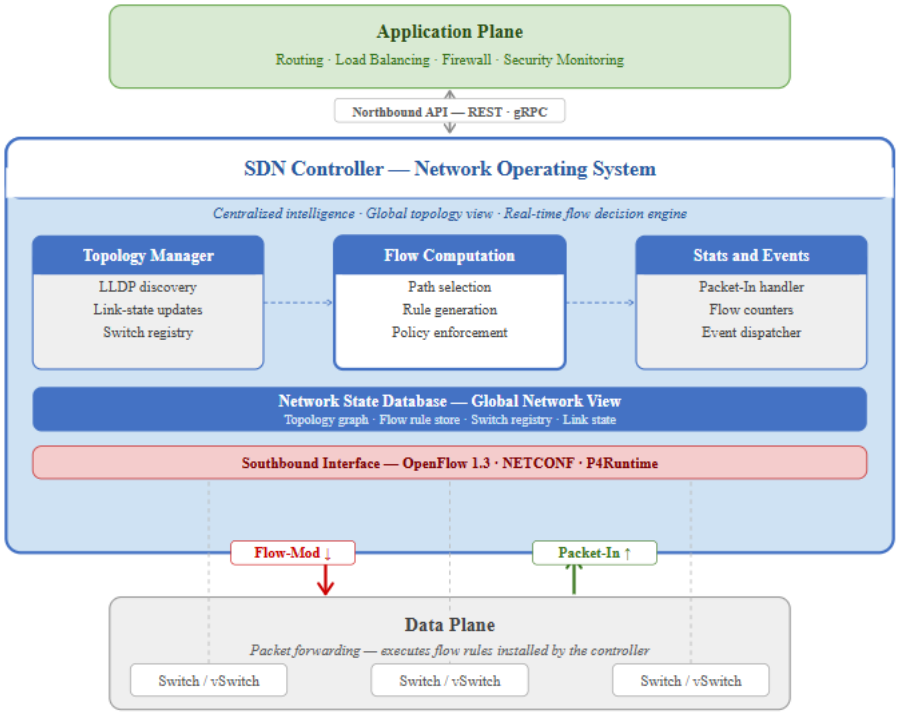


Figure 1.2 – Internal Architecture of the SDN Control Plane and its Interactions with the Application and Data Planes.

Internally, the controller maintains a network graph G = (V, E), where V represents the set of network switches and E the set of active links, discovered and updated continuously through topology protocols such as LLDP. This global view enables the controller to make informed forwarding decisions for any source-destination pair in the network. When a switch encounters a packet that matches no existing flow rule a table-miss it encapsulates the packet header into a Packet-In message and forwards it to the controller. The controller then resolves the destination, computes the optimal forwarding path, and reactively installs flow rules across all switches along that path through Flow-Mod messages. Algorithm 1.1 formalizes this reactive flow installation process.

**Algorithm 1.1: Reactive Flow Rule Installation (based on [12][13])**

```
Input: Packet-In message from switch s, carrying packet P
Output: Flow rules installed across data plane switches
H ← extract_headers(P)                    // L2/L3/L4 fields
Update topology graph G if topology event detected
src ← s                                   // ingress switch
dst ← resolve_destination(H, G)   // lookup destination
switch
if dst = unknown then
   flood(P); return                // destination not yet
learned
path ← shortest_path(G, src, dst)   // e.g., Dijkstra / BFS
```

```
8:  for each switch u_i along path do
9:    match ← build_match(H)          // construct match fields
10:   action ← output(port(u_i → u_i+1))  // determine output port
11:   rule ← ⟨match, priority, action, timeout⟩
12:   send Flow-Mod(u_i, rule)   // install rule via southbound
API
13:  send Packet-Out(s, P, port(s → next_hop))
// forward buffered packet
```

The path computed in line 7 minimizes hop count by default, but can be substituted with any policy-driven metric bandwidth, latency, or energy consumption depending on the application plane objective [14][15]. This computation operates with a time complexity of O(V + E), where V denotes the number of network switches and E the number of active links in G. Under normal conditions this overhead is negligible; however, as network scale increases, the cumulative cost of repeated path computations across concurrent Packet-In events becomes a measurable factor in controller processing latency.

Under DDoS or scanning attack conditions, this dependency on G introduces an additional vulnerability. Traffic flooding generates a sustained stream of Packet-In messages that congests the southbound channel and exhausts controller resources, delaying LLDP-based topology updates and causing path computations to operate over a stale or corrupted network view [16][17]. This directly motivates the need for attack detection prior to forwarding decision, a principle underlying the design presented in Chapter 3.

Once flow rules are installed, subsequent packets belonging to the same flow are forwarded directly by the data plane without any controller involvement, demonstrating the fundamental principle of SDN: intelligence resides in the control plane, while speed resides in the data plane [6][11].

### 1.2.1.3. Application Plane

The application plane is the topmost layer of the SDN architecture and serves as the policy abstraction layer of the network. Rather than enforcing decisions, it defines what the network should achieve, expressing high-level objectives and constraints that the control plane then translates into concrete forwarding rules installed across the data plane. It is the layer where network behavior is defined not through low-level device configuration, but through software applications that express high-level policies and objectives, which are then translated by the control plane into concrete forwarding rules installed across the data plane [4][5]. This separation of intent from implementation is what makes SDN fundamentally different from all prior networking paradigms [6][18].

Applications in this plane interact with the SDN controller exclusively through the northbound API reading the global network state SS S maintained by the controller, and writing back policy decisions that trigger flow rule computation and installation [13][18]. Each application is therefore a policy engine: it observes the network, computes an objective, and instructs the controller what to enforce.

The practical scope of the application plane is broad. Google's B4 [18] one of the most influential real-world SDN deployments implements a traffic engineering application that continuously solves Equation 1.3 to maximize link utilization across a globally distributed WAN, achieving near-100% utilization on intercontinental links that previously operated at 30~40%. Beyond traffic engineering, application plane functions span access control and security policy enforcement, network virtualization and slicing, fault detection and self-healing, and QoS management [5][19]. All of these applications operate on the same global view provided by the controller, without any direct interaction with physical devices a stark contrast to legacy networks where each such function required per-device configuration through vendor-specific interfaces [6][13].

### 1.2.2. Southbound and Northbound APIs (OpenFlow, P4)

The decoupling of the control and data planes in SDN is only achievable through well-defined programmatic interfaces that govern communication between the layers of the architecture. These interfaces the southbound API between the controller and the data plane, and the northbound API between the controller and the application plane are the connective tissue of the SDN model [5][13]. Without them, the separation of planes would remain a conceptual abstraction rather than an operational reality.

#### 1.2.2.1. Southbound API

The southbound API is the interface through which the SDN controller installs, modifies, and deletes flow rules in the data plane devices, and through which those devices report events and statistics back to the controller [6][13]. It is the most critical interface in the SDN architecture its expressiveness and efficiency directly determine what the controller can enforce and how quickly it can respond to network events [5].

The southbound interface operates through two primary message types. The controller sends Flow-Mod messages to install or update forwarding rules, formally defined as:

$$FlowMod = \langle m, pr, A, t, flags \rangle \tag{1.3}$$

where m is the match pattern, pr the priority, A the instruction set, t the timeout parameters, and flags encode the operation type add, modify, or delete [7]. The data plane responds with Packet-In messages upon table-miss events, carrying the unmatched packet to the controller for reactive policy decisions [7][8].

Two protocols have come to define the southbound API landscape OpenFlow and P4 representing two fundamentally different philosophies of data plane programmability.

OpenFlow, introduced by McKeown et al. [6] and standardized by the Open Networking Foundation [7], operates on a fixed pipeline model. The forwarding behavior of the switch is constrained to a predefined set of header fields and actions specified by the protocol version. The controller programs the switch by populating its flow tables within these fixed boundaries.

OpenFlow's strength is its simplicity and broad hardware support; its limitation is that the pipeline cannot be extended beyond what the specification defines [9].

P4 (Programming Protocol-Independent Packet Processors), proposed by Bosshart et al. [9], takes a fundamentally different approach a programmable pipeline model. Rather than conforming to a fixed set of headers and tables, P4 allows the operator to define the parser, the match-action tables, and the control flow of the pipeline itself. The P4 parser is formally modeled as a finite state machine:

$$\mathcal{M} = (Q, \Sigma, \delta, q_0, F) \tag{1.4}$$

where Q is the set of parser states, Σ is the set of recognizable header types, δ: Q × Σ → Q is the state transition function, $q_0$ is the initial state, and F ⊆ Q is the set of accept states [9][20]. This formulation gives P4 programs the ability to parse arbitrary protocol headers including custom, proprietary, and future protocols not yet standardized without any modification to the underlying hardware pipeline. As a result, P4 is significantly more expressive than OpenFlow, which is constrained to a fixed set of predefined header fields and actions. This expressiveness comes at the cost of greater implementation complexity: P4 programs must be compiled into target-specific binaries for each hardware platform, whereas OpenFlow rules are pushed directly to the switch at runtime. Table 1.1 summarizes the principal differences between the two protocols across the dimensions most relevant to SDN deployment

Table 1.1 – OpenFlow vs P4: Southbound API Comparison (adapted from [9][13]).

| **Feature** | **OpenFlow** | **P4** |
|---|---|---|
| Pipeline model | Fixed predefined by protocol specification | Programmable user-defined parser and control flow |
| Header parsing | Fixed set of 44 standardized match fields | Arbitrary parser defined as FSM |
| Match fields | Spec-constrained (Ethernet, IP, TCP/UDP, VLAN, MPLS) | Protocol-independent any user-defined header |
| Action set | Predefined (output, drop, push/pop VLAN, modify-field) | User-defined including custom stateful operations |
| Table-miss behavior | Packet-In to controller incurs controller RTT | User-defined miss action handled within pipeline |
| Processing latency on miss | Controller round-trip millisecond range | No controller involvement nanosecond range |
| Reconfigurability | Flow rules modifiable at runtime without recompilation | Pipeline changes require recompilation and target reload |
| Telemetry | Per-flow counters via stats request/reply cycle | Native In-band Network Telemetry (INT) per-packet |
| Compilation target | Controller pushes Flow-Mod directly to switch | P4 compiler generates target-specific binary (ASIC, FPGA, SmartNIC) |

#### 1.2.2.2. Northbound API

The northbound API is the interface through which network applications communicate with the SDN controller. While the southbound API solves the problem of how the controller talks to switches, the northbound API solves a fundamentally different problem how applications express network objectives without needing to understand the underlying physical topology, device capabilities, or protocol details [5][13]. It is the interface that elevates SDN from a programmable forwarding fabric to a fully programmable network platform.

Unlike the southbound API, which is governed by well-defined open standards such as OpenFlow and P4, the northbound API has no universal specification [5][21]. Each controller platform exposes its own northbound interface ranging from low-level RESTful and gRPC APIs that give applications direct read and write access to the controller's internal network state, to high-level intent-based abstractions where the application declares only what it wants the network to achieve and leaves the path computation, rule generation, and installation entirely to the controller [13][22]. This progression from imperative to declarative interaction is what gives the northbound API its architectural depth and is precisely the abstraction that makes the application plane's policy optimization Equation 1.3 operationally meaningful [16][22].

The absence of a standardized northbound API remains one of the most significant open challenges in SDN, as it creates tight coupling between applications and specific controller implementations, severely limiting portability across platforms [5][21]. Controllers such as ONOS and OpenDaylight have made concrete progress toward addressing this through intent frameworks that expose a controller-agnostic northbound abstraction layer but a universally accepted standard remains an open problem in the SDN research community [21][22].

## 1.3. Distributed Denial of Service (DDoS) Attacks in SDN

### 1.3.1. DDoS Attack Mechanisms and Taxonomy

#### 1.3.1.1. Definition

A Distributed Denial-of-Service (DDoS) attack is a coordinated attempt by multiple compromised hosts to exhaust the resources of a target system computational, memory, or network bandwidth to the point where legitimate service requests can no longer be fulfilled. Unlike single-source DoS attacks, the distributed nature of DDoS campaigns makes source filtering insufficient as a primary defense: attack traffic originates from geographically dispersed nodes, often indistinguishable from legitimate clients at the packet level [23]. Figure 1.3 illustrates this attack model in an SDN environment, where a botnet under attacker control directs coordinated traffic through the data plane bypassing the SDN controller's visibility until the target server is rendered unavailable.

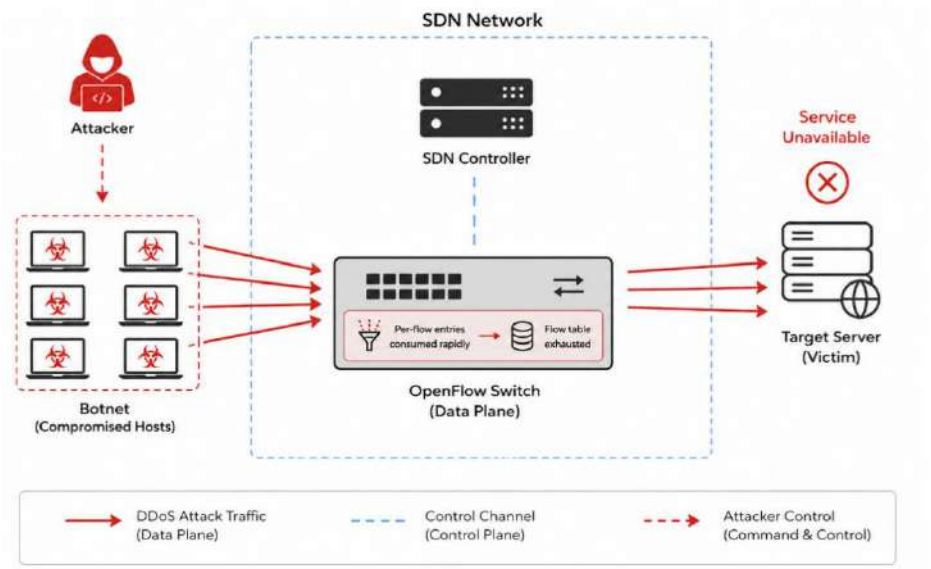


Figure 1.3 – DDoS attack scenario in an SDN network. Botnet traffic is forwarded through the OpenFlow switch toward the target, resulting in service unavailability.

The aggregate bandwidth generated by such an attack can be expressed as:

$$B = N.r.L \tag{1.5}$$

where B is the total attack bandwidth in bits per second, N is the number of participating botnet nodes, r is the per-host packet rate in packets per second, and L is the mean packet size in bytes [23]. Even modest botnets on the order of $10^3$ to $10^4$ nodes can sustain aggregate traffic volumes sufficient to saturate enterprise-grade uplinks, demonstrating that the threat does not require large-scale infrastructure to be operationally devastating.

### 1.3.1.2. Taxonomy

Defending against DDoS attacks requires knowing precisely what you are defending against. The term "DDoS" encompasses a wide range of attack strategies that differ not just in technique but in the fundamental resource they target and the way they consume it. Treating them as a single class leads to defense architectures that are either over-engineered for threats they will never face or blind to the ones they will. Classification is therefore not background context it is the first design decision.

The first axis of classification is the protocol layer targeted. An attack can exhaust resources at the network layer (L3) by flooding links with raw traffic volume, at the transport layer (L4) by saturating stateful connection handling, or at the application layer (L7) by exploiting the computational cost of legitimate request processing. This distinction matters because the layer determines where in the network stack the damage occurs and consequently where detection and mitigation must intervene.

The second axis is the exhaustion strategy. Volumetric attacks generate traffic at a scale that exceeds the victim's capacity damage is proportional to volume generated. Asymmetric attacks exploit the disparity between the cost of issuing a request and the cost of fulfilling it: the attacker spends little, the victim spends much. This distinction determines whether volume is a reliable detection signal at all. Against a volumetric attack, aggregate traffic anomaly is a sufficient detection basis. Against an asymmetric attack, traffic may appear entirely normal in volume while the server is being systematically exhausted rendering volume-based detection not merely insufficient but actively misleading.

These two axes are orthogonal, and Table 1.2 maps the principal attack classes across both.

Table 1.2 – DDoS Attack Taxonomy by Protocol Layer and Exhaustion Strategy.

| **Attack Type** | **Protocol Layer** | **Exhaustion Strategy** | **Primary Resource Targeted** | **Amplification Potential** |
|---|---|---|---|---|
| UDP Flood | Network (L3/L4) | Volumetric | Bandwidth | Low (direct)/High (reflected) |
| ICMP Flood (Smurf) | Network (L3) | Volumetric | Bandwidth | High (broadcast amplification) |
| TCP SYN Flood | Transport (L4) | State exhaustion | Connection table | Low |
| HTTP Flood | Application (L7) | Asymmetric (request cost) | CPU/thread pool | Medium |
| Slow-rate (Slowloris) | Application (L7) | Asymmetric (session holding) | Connection slots | Medium |

What the table makes immediately visible is that the five attack classes form two structurally distinct groups separated by both axes simultaneously. UDP flood, ICMP flood, and TCP SYN flood all operate below L7 and produce detectable deviations in aggregate traffic behavior. HTTP flood and Slowloris both operate at L7 and remain within normal traffic volume bounds while causing service degradation. No single detection primitive straddles this boundary cleanly.

For the volumetric group, the critical quantity is the aggregate bandwidth the attack can generate. In reflected and amplified variants, the attacker spoofs the victim's IP as the source of requests directed at open resolvers or broadcast domains, causing third-party infrastructure to flood the victim with responses.

For the asymmetric group, volume is the wrong signal entirely. HTTP flood issues valid, complete requests; Slowloris holds connections open with deliberately incomplete headers. Both are individually indistinguishable from legitimate traffic at the packet level. Detection requires observing statistical patterns across flows over time request rate distributions, connection duration profiles, response-to-request ratios rather than inspecting any individual packet [24][25].

The escalating frequency of DDoS attacks over recent years underscores the urgency of developing efficient detection mechanisms. According to NETSCOUT ATLAS Threat Intelligence Reports, global DDoS attack frequency rose from 10.09 million in 2020 to

approximately 15.8 million in 2023, with H1 2023 alone recording 7.9 million attacks, a 31% year-over-year increase [26]. Figure 1.4 illustrates this growth trend.

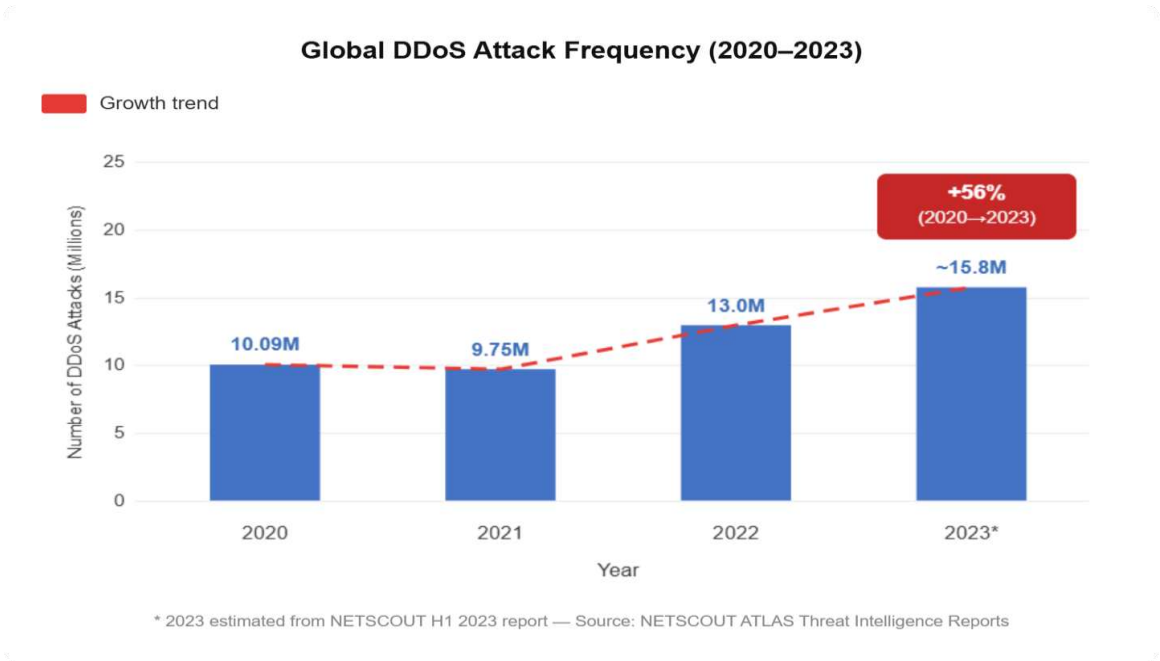


Figure 1.4 – Global DDoS Attack Trends (2020–2023).

### 1.3.2. Impact of DDoS Attacks on SDN Architecture

While Software-Defined Networking (SDN) revolutionizes network management by decoupling the control plane from the data plane, this centralized design introduces significant security vulnerabilities. Distributed Denial of Service (DDoS) attacks against SDN environments are particularly devastating as they exploit the inherent communication dynamics between decentralized switches and the centralized controller [27].

The impact of a DDoS attack on an SDN architecture can be categorized into three primary vectors:

**Data Plane Impact: Flow Table Exhaustion**

In the data plane, SDN relies on OpenFlow switches to forward traffic based on flow rules stored in Ternary Content Addressable Memory (TCAM). However, TCAM is expensive and power-intensive, typically accommodating only a few thousand flow entries [28],[29]. During a DDoS attack, an adversary generates a massive volume of packets with randomized, spoofed IP addresses. Each unique packet creates a "table-miss," forcing the switch to request a new rule. As shown in Figure 1.5(a), this rapidly fills the limited TCAM capacity, causing Flow Table Exhaustion. Once the table hits 100% capacity, the switch must drop legitimate packets or overwrite valid rules, leading to total network unavailability [29].

**Control Plane Impact: Controller Starvation**

The centralized controller acts as the "brain" of the SDN. When a switch encounters a table-miss, it encapsulates the packet header into a Packet-In message and forwards it to the controller [30]. In a flooding attack, this flood of requests forces the controller to continuously calculate paths and issue Flow-Mod messages. As illustrated in Figure 1.5(b), the CPU load increases exponentially with the packet rate. Once the controller hits 100% CPU utilization, it enters a state of Controller Starvation [27],[30]. If the controller becomes unresponsive, it creates a Single Point of Failure (SPOF) that paralyzes the entire network.

**Communication Channel Impact: Southbound Interface Saturation**

The Southbound API (typically OpenFlow) is the dedicated link bridging the control and data planes. The barrage of Packet-In requests traveling up to the controller, combined with the Flow-Mod messages traveling down, creates a massive bottleneck. This saturates the link bandwidth, increasing network latency and preventing critical control traffic from being processed [27][28].

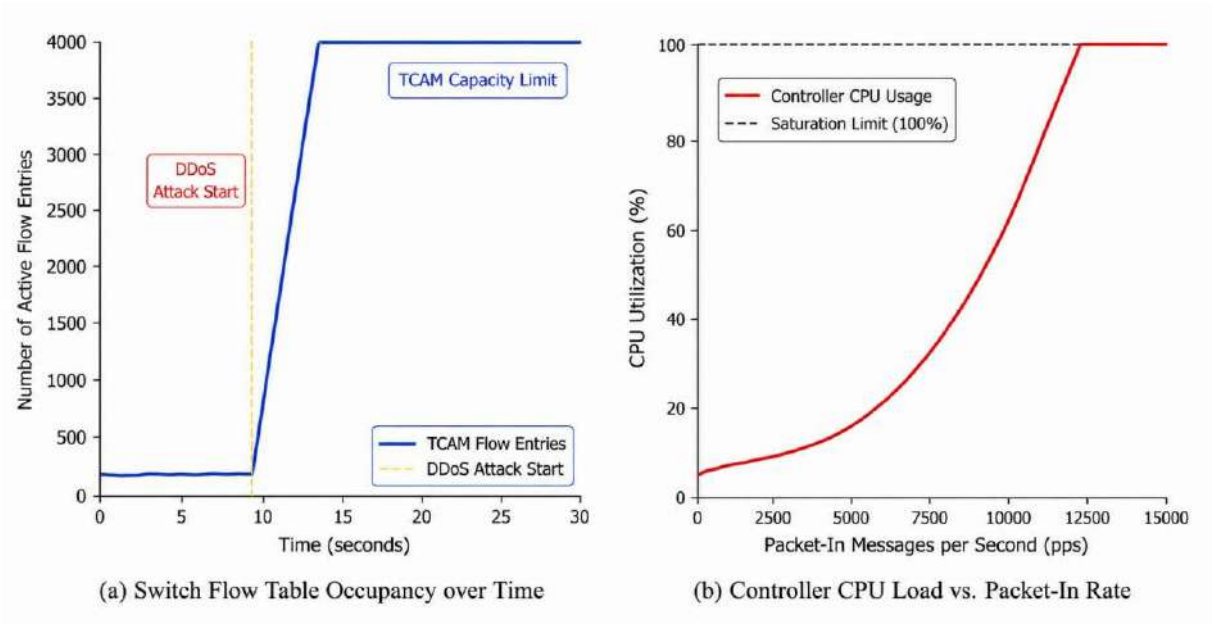


Figure 1.5 – Resource exhaustion during a DDoS attack. (a) Displays the rapid saturation of switch TCAM; (b) Illustrates the exponential increase in controller CPU utilization.

## 1.3.3. Horizontal and Vertical Scanning

**Horizontal scanning** probes one fixed port across many IP addresses (e.g., TCP/445 on $H_1$, $H_2$, $H_n$). It is the signature of worm propagation and mass-exploitation campaigns, one vulnerability, many targets [31].

**Vertical scanning** probes many ports on a single target host (ports 1–65535 on one IP). It characterizes targeted pre-exploitation reconnaissance, mapping a specific victim's attack surface before selecting a vector [31].

**Block (subnet) scanning** combines both: multiple ports across a subnet, typically distributed across botnet nodes to stay below per-source detection thresholds [32]. Each type produces a distinct traffic matrix and demands a different detector. Horizontal scans are caught by destination fan-out anomalies (one source, many IPs, same port). Vertical scans are identified by per-destination port-entropy (many ports, one IP). Block scans require cross-source aggregation [33].

Figure 1.6 illustrates this distinction within an SDN domain, showing a scanner outside the network boundary directing probes toward edge switches. The red dashed arrows represent scan probes arriving at both switches simultaneously, triggering Packet-In floods toward the controller through the OpenFlow channel. The two failure points are visible: TCAM exhaustion at the data plane switches and CPU saturation at the controller both caused by a single attacker operating outside the domain.

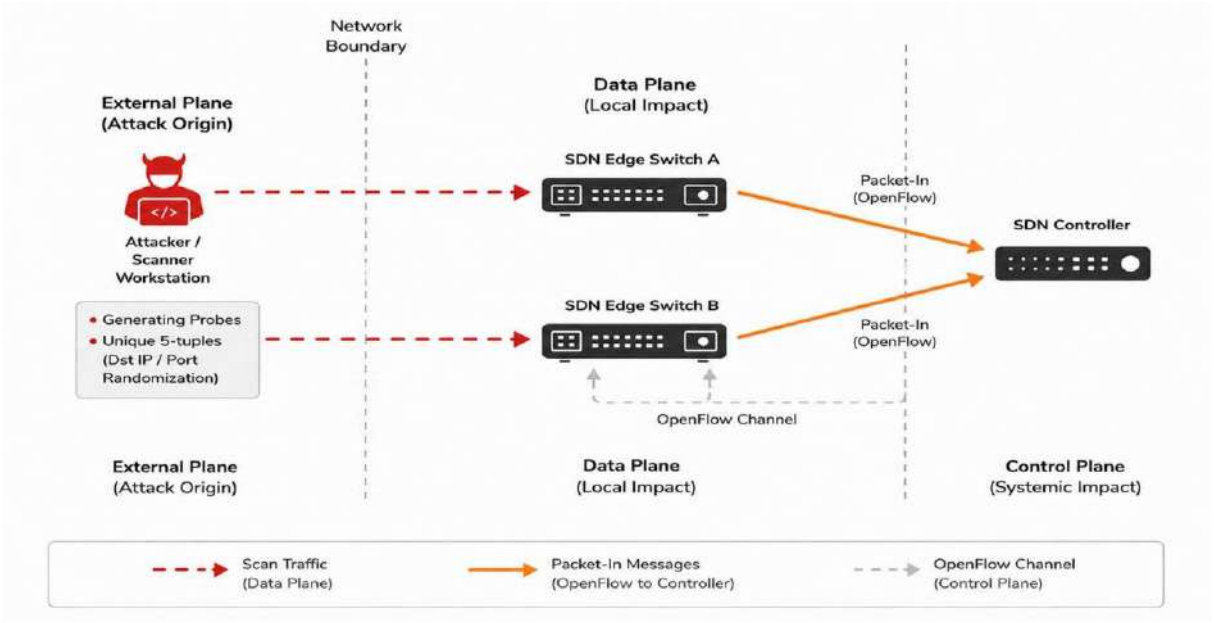


Figure 1.6 – Scanning Attack Scenario in SDN: From Probe Generation to Controller Notification.

Network scanning activity has followed a similarly alarming trajectory. According to F5 Labs Sensor Intelligence Series reports, which aggregate data from globally distributed honeypot sensors, observed scanning events increased from 5.5 million in 2022 to 8.7 million in 2024, representing a 58% increase over this period [34]. Figure 1.7 illustrates this evolution, confirming that reconnaissance-based threats are becoming increasingly prevalent in modern network environments.

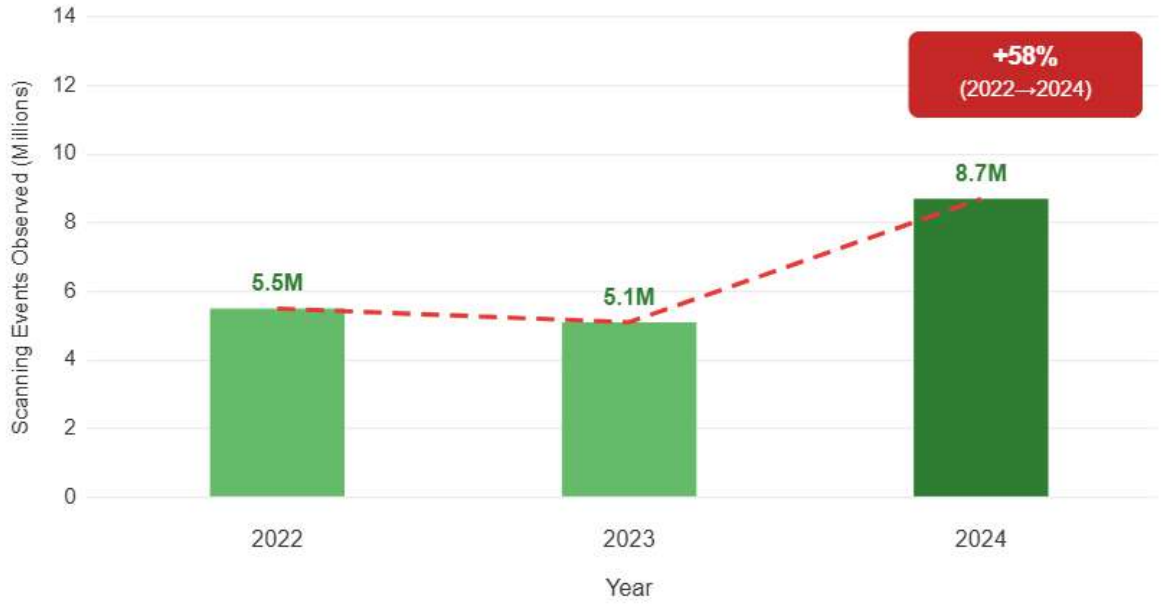


Figure 1.7 – Global Network Scanning Trends (2020–2024)

### 1.3.4. Flow-Level Behavior and Impact on SDN

In OpenFlow-based SDN, a packet with no matching flow entry triggers a **table-miss**, generating a *Packet-In* message to the centralized controller for rule computation and installation [32][35]. Scanning attacks structurally abuse this mechanism: each probe targets a new destination IP or port, producing a unique 5-tuple that guarantees a table-miss on every single packet an uninterrupted stream of first-seen flows each demanding separate controller intervention [32].

The consequences propagate across two planes. At the control plane, even 1,000 probes per second is sufficient to saturate controller CPU, stalling legitimate flow installation network-wide [34]. At the data plane, the resulting rule installations fill the TCAM with thousands of single-use entries until exhaustion forces packet drops or rule eviction, degrading all other flows [37]. In legacy networks a scan stresses the scanned host; in SDN it stresses the controller and therefore every host sharing it [37][38][39].

At the flow level, scanning is distinguishable from normal traffic and DDoS through two signatures: an abnormally high new-flow ratio, and a divergent entropy profile horizontal scanning collapses destination IP entropy while port entropy stays high; vertical scanning inverts this; DDoS collapses both simultaneously [32][33]. This three-way divergence is the statistical foundation of the detection mechanism proposed in Chapter 3.

Figure 1.8 quantifies this impact, plotting Packet-In rate and controller CPU load as a function of scan rate. Both metrics rise sharply beyond 250 probes per second, with CPU reaching full saturation at 1,000 probes per second beyond which the Packet-In backlog continues to grow while the controller can no longer respond to legitimate flows.

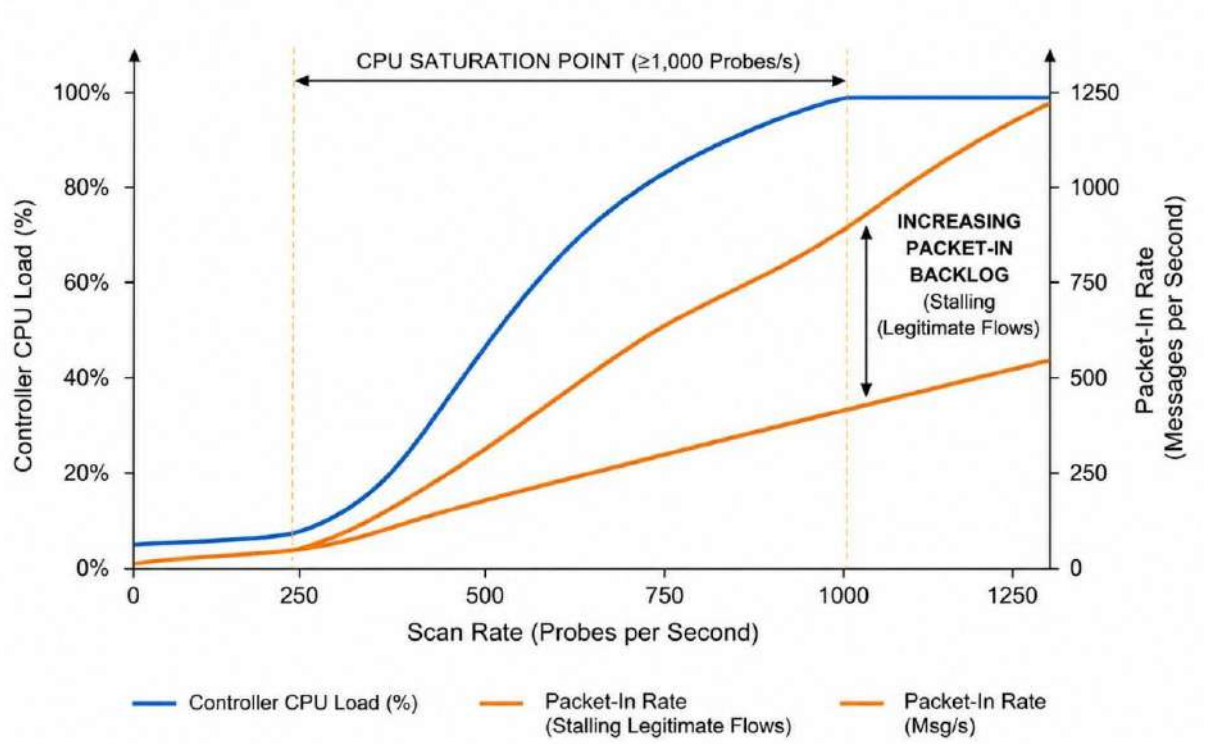


Figure 1.8 – SDN Controller Resource Exhaustion Under Scanning Attack Stress.

## 1.4. Conclusion

This chapter has established the architectural and security foundations underlying this thesis. The SDN architecture, built around the separation of control and data planes and the centralization of decision-making in the controller, offers significant advantages in terms of programmability and network management. However, this same centralization introduces a structural vulnerability: both DDoS and scanning attacks exploit the controller's dependency on Packet-In processing and flow table management to exhaust its resources, degrading or paralyzing the entire network.

This analysis highlights a fundamental tension the mechanisms that make SDN powerful are the same ones that make it vulnerable. Addressing this tension requires detection approaches that are lightweight enough to operate within the controller's constraints while remaining accurate enough to distinguish attack traffic from normal behavior. The state of the art in such approaches is reviewed in Chapter 2.

Chapter

2

# State of the Art: Intrusion Detection Mechanisms in Software-Defined Networks

## 2.1 Introduction

The centralization of the SDN control plane introduces a powerful network management paradigm, but also exposes the controller to a range of security threats that conventional network architectures were not designed to address. As intrusion detection has become a critical requirement in SDN deployments, a broad body of research has emerged proposing detection mechanisms that vary significantly in their underlying techniques, computational requirements, and suitability for real-time operation.

This chapter surveys the state of the art in intrusion detection for SDN environments. It examines data-driven approaches based on classical machine learning and deep learning, analyzes lightweight statistical and rule-based alternatives, and reviews feature selection strategies relevant to constrained deployment contexts. Each category is assessed with respect to its detection capability, computational overhead, and practical applicability within the SDN controller architecture.

The chapter concludes by identifying the key limitations of existing work and positioning the research gap that motivates the lightweight detection and mitigation approach proposed in Chapter 3.

## 2.2 Detection Approaches for Network Security

The identification of malicious traffic in Software-Defined Networking settings has been addressed using a wide array of techniques, from resource-intensive machine learning models to efficient statistical methods. The selection of the detection method is significantly affected by the operational limitations of the deployment environment, especially in SDN, where the controller serves as a centralized and resource-constrained entity. This section analyzes the two primary kinds of detection methodologies identified in the literature, assessing their relevance, advantages, and constraints within the framework of SDN-based security.

### 2.2.1. Machine Learning and Deep Learning Based Detection

As depicted in Figure 2.1, data-driven intrusion detection methods fall into two broad categories: classical machine learning, which requires explicit feature engineering, and deep learning, which extracts representations automatically from raw traffic data. The two families present fundamentally different trade-offs between detection performance and computational cost.

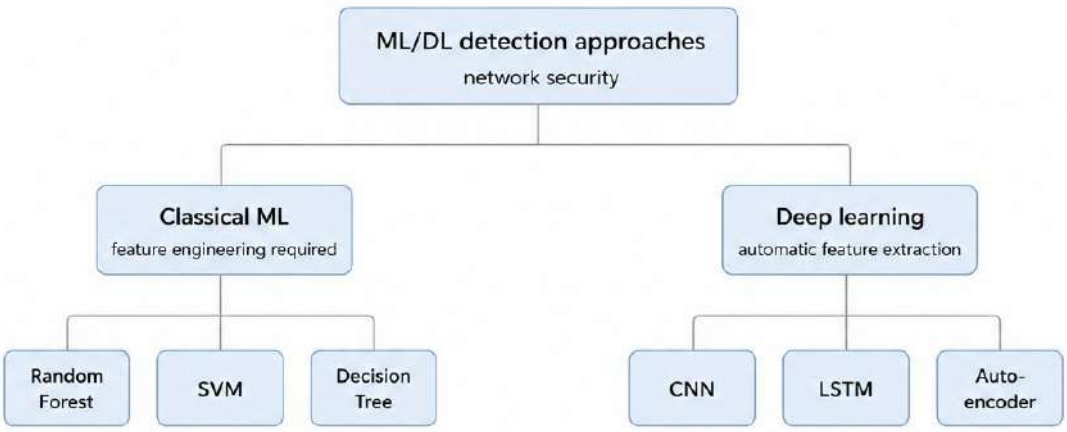


Figure 2.1 – Taxonomy of ML/DL Detection Approaches and Their Computational Overhead.

#### 2.2.1.1. Classical Machine Learning

The earliest data-driven intrusion detection systems were built around classical machine learning algorithms. A comprehensive survey of data mining and machine learning methods for cyber security established Random Forest, Support Vector Machines, and Decision Trees as the most widely adopted paradigms in this domain, providing a systematic taxonomy of their applicability across different attack categories [40]. A comparative study further evaluating these methods across the KDD CUP99 and NSL-KDD datasets concluded that ensemble learning approaches generally yield superior detection accuracy, while also highlighting the distinct trade-offs each algorithm presents depending on the nature of the traffic and the attack type encountered [41]. A dedicated evaluation of Random Forest and SVM demonstrated that combining both methods

through careful feature selection can further improve classification performance on network intrusion tasks [42].

These methods share a common dependency on hand-crafted feature vectors derived from flow-level measurements such as packet rates, byte counts, inter-arrival times, and protocol flag distributions a requirement that introduces fragility when traffic behavior shifts over time and periodic retraining becomes necessary.

#### 2.2.1.2. Deep Learning Architectures

Deep learning introduced a fundamentally different approach by eliminating the need for manual feature engineering. A deep hierarchical network applied directly to raw flow data demonstrated that automatic feature extraction through successive nonlinear transformations can achieve competitive intrusion detection performance without relying on pre-engineered statistics [43].

Among recurrent architectures, LSTMs and GRUs have proven particularly well suited to sequential traffic analysis. Radford et al. demonstrated that recurrent neural networks can effectively model temporal dependencies in network traffic for anomaly detection, capturing behavioral patterns that unfold across time rather than within individual packets [44]. This capability was further reinforced by a GRU-based model augmented with optimized feature selection, which showed measurable gains in detection accuracy over baseline recurrent models [45].

CNNs have been applied to intrusion detection by exploiting their ability to extract local feature correlations across traffic attributes. An evaluation of CNN architectures across the KDDCup 99 and NSL-KDD datasets demonstrated their capacity to learn discriminative spatial patterns from raw traffic representations, with performance comparable to recurrent models on several attack categories [46]. Autoencoders represent a distinct branch operating in an unsupervised manner, learning a compact representation of normal traffic and flagging deviations through elevated reconstruction error at inference time. An ensemble-based autoencoder system demonstrated that aggregating multiple autoencoders trained on different feature subsets produces more robust anomaly scores than a single monolithic model [47].

#### 2.2.1.3. Lightweight Variants

The computational demands of deep learning have motivated a dedicated line of research into lightweight architectures suitable for resource-constrained deployment. A comprehensive survey on lightweight deep learning examined the design principles including model compression, pruning, and efficient architecture search that enable capable models to operate within strict memory and processing budgets [48]. These findings establish that the boundary between detection capability and deployment efficiency is not fixed, and that architectural optimization can bring deep learning intrusion detection within reach of constrained network infrastructure.

Despite the detection capabilities offered by both classical and deep learning approaches, the computational overhead they introduce under real-time conditions motivates the exploration of lighter and training-free alternatives, examined in the following section.

#### 2.2.1.4. Limitations in SDN Deployment Contexts

Despite their strong detection performance, machine learning and deep learning approaches share a set of structural limitations that constrain their applicability within SDN controller environments. First, all supervised learning methods require labeled training datasets that accurately represent both normal and attack traffic distributions, a requirement that is difficult to satisfy given the dynamic and heterogeneous nature of real SDN deployments [22][23]. Second, trained models require periodic retraining as traffic behavior evolves, introducing operational overhead that conflicts with the continuous availability requirement of the SDN controller. Third, and most critically, the inference computational cost of complex models competes directly with the controller's primary function of flow rule computation and installation, risking performance degradation under the very attack conditions they are designed to detect [20][21]. Recent systematic reviews confirm that computational overhead and real-time constraint remain the two most consistently identified unresolved challenges in ML-based intrusion detection for SDN environments [22][33].

These limitations motivate the exploration of lightweight, training-free detection approaches examined in the following section.

Table 2.1 quantifies these overhead constraints across the principal ML and DL architectures reviewed in this section, comparing their inference complexity, parameter count, and suitability for real-time SDN controller deployment.

Table 2.1 – Computational Complexity of ML/DL Approaches per Detection Cycle (compiled from standard algorithmic analysis).

| **Model** | **Inference Complexity** | **Parameters** | **Real-Time SDN Viable** |
|---|---|---|---|
| Random Forest | O(F.D.T) | O(D.T) | No, scales with trees and features |
| SVM | O(F.SV) | O(SV) | No, support vector count grows with data |
| CNM | $O(F.D.K^2)$ | Millions | No, convolutional layers require GPU |
| LSTM | $O(F.D.H^2)$ | Millions | No, recurrent state requires sequential processing |
| Autoencoder | O(F.D.L) | Millions | No, reconstruction cost scales with layers |
| EWMA+Rules (proposed) | O(F) | O(S) | Yes, constant per source, no model storage |

F = active flow entries, D = feature dimensions, T = number of trees, SV = support vectors, K = kernel size, H = hidden state size, L = number of layers, S = number of active sources.

## 2.3 Lightweight Anomaly Detection Methods

Although machine learning and deep learning techniques exhibit high detection accuracy, their reliance on training data and the computational burden they impose render them challenging to maintain consistently in resource-limited settings. A complementary set of approaches tackles this gap by directly utilizing the statistical characteristics of network traffic, eliminating the need for a preliminary training process. These lightweight anomaly detection methods encompassing entropy-based metrics, statistical control techniques, rule-based adaptive threshold mechanisms, and principled feature selection strategies provide a more feasible deployment framework in environments where detection must operate concurrently with other time-critical processing activities.

### 2.3.1. Entropy-Based Detection

Entropy-based detection exploits the statistical regularity of normal network traffic to identify anomalous behavior. Under normal conditions, traffic is distributed across a wide range of source addresses, destination ports, and protocol types, producing consistently high entropy values. This regularity breaks down characteristically under attack: volumetric floods compress destination distributions, scanning campaigns unnaturally broaden port ranges, and spoofing attacks distort source address diversity. In each case, the entropy of one or more traffic features deviates measurably from its baseline, producing a detection signal without requiring any prior model training [49].

The foundational measure used in this context is Shannon entropy, defined as:

$$H(x) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{2.1}$$

where $p(x_i)$ represents the probability of observing traffic feature $x_i$ within a given time window. A significant and sustained decrease in H(X) below a predefined threshold indicates a concentration of traffic consistent with an attack pattern.

In the context of network traffic analysis, entropy measures the diversity of observed values within a given traffic feature over a fixed time window. Under normal conditions, traffic is distributed across a wide range of source IP addresses, destination ports, and protocol types, resulting in high entropy values. When an attack occurs, this distribution becomes skewed in a characteristic way depending on the attack type: a port scanning attempt concentrates traffic across many destination ports from a single source, a DDoS flood concentrates packets toward a single destination, and an IP spoofing campaign produces an unnaturally uniform distribution of source addresses. In each case, the entropy of the affected traffic feature deviates measurably from its baseline, producing either a sharp drop or an anomalous spike that serves as a detection signal [49]. Rather than inspecting individual packets, the method monitors the statistical distribution of traffic features over time and raises an alert when entropy crosses a predefined

threshold. This behavior is illustrated in Figure 2.2, which shows H(X) remaining stable during normal traffic before dropping sharply below the detection threshold at the onset of an attack.

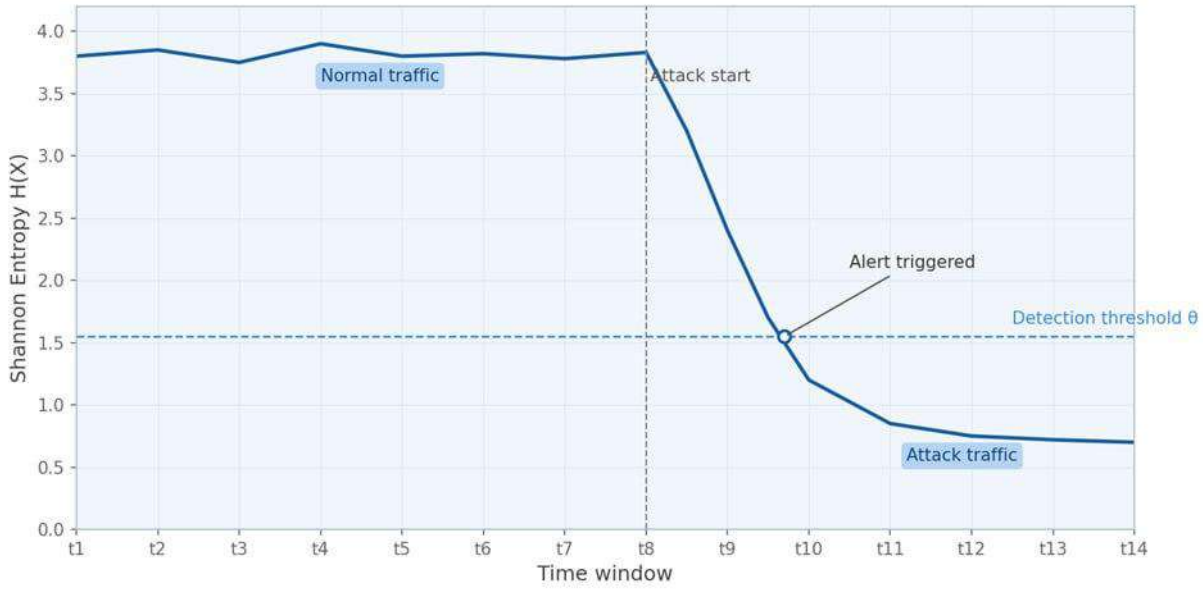


Figure 2.2 – Shannon entropy H(X) under normal and attack traffic conditions.

In SDN environments, this mechanism maps naturally onto the controller-switch architecture. Entropy can be computed locally at each switch using flow-level counters already maintained by OpenFlow, and the resulting value is reported to the controller only when a threshold violation is detected, avoiding unnecessary control plane traffic under normal conditions [49]. When entropy is integrated directly into the OpenFlow pipeline, the detection process can trigger immediate flow rule modifications such as dropping or redirecting suspicious flows without requiring any external processing or round-trip communication with the controller [50]. This tight coupling between measurement and response reduces both detection latency and control plane overhead, making entropy a particularly suitable mechanism for lightweight anomaly detection in SDN deployments [51].

### 2.3.2. Statistical Detection Methods

While entropy captures distributional shifts in traffic features, a broader class of statistical methods addresses anomaly detection by modeling the temporal evolution of traffic metrics and flagging deviations that exceed statistically defined boundaries. Two control chart techniques have received particular attention in this context: the Cumulative Sum (CUSUM) and the Exponentially Weighted Moving Average (EWMA).

CUSUM operates by accumulating deviations from an expected baseline over time. Rather than reacting to a single anomalous observation, it builds evidence across consecutive measurements, making it sensitive to persistent but subtle shifts that would go undetected by instantaneous threshold comparisons. The statistic is updated at each time step as:

$$S_t = \max\{0, s_{t-1} + (x_t - \mu_0) - k\} \quad (2.2)$$

where $x_t$ is the observed traffic metric at time t, $\mu_0$ is the expected mean under normal conditions, and k is a slack parameter that controls sensitivity to small deviations. The max {} operator prevents the statistic from accumulating negative values, effectively resetting the evidence counter when traffic returns to normal. An alert is raised when $S_t$ exceeds a predefined threshold h, indicating that traffic has persistently shifted beyond acceptable bounds. This accumulation property makes CUSUM particularly effective against slow-rate attacks that gradually escalate across multiple time windows rather than manifesting as a single sudden spike.

EWMA takes a different approach by assigning exponentially decreasing weights to past observations, producing a smoothed statistic that reflects recent traffic behavior more strongly than older history. The statistic is defined as:

$$z_t = \lambda x_i + (1 - \lambda) z_{t-1} \quad (2.3)$$

where $\lambda \epsilon$ (0,1] is a smoothing parameter. Expanding this recursion reveals that $z_t$ is a weighted sum of all past observations, with weights decaying geometrically as $\lambda(1-\lambda)^j$ for an observation j step in the past. An anomaly is declared when $z_t$ crosses a control limit defined as:

$$\frac{UCL}{LCL} = \mu_0 \pm \kappa . \sigma \sqrt{\frac{\lambda}{2 - \lambda}} \quad (2.4)$$

where sigma is the standard deviation of the traffic metric under normal conditions and k is a sensitivity multiplier. As lambda decreases, the control limits narrow, producing a smoother statistic less reactive to transient spikes. Conversely, larger values of lambda widen the memory window and increase responsiveness to abrupt changes. Ye et al. demonstrated that under normal system activity EWMA forecast errors behave as a white noise process, and that intrusive behavior produces a statistically detectable departure from this property, confirming its applicability to computer intrusion detection [52]. In online deployments, sigma is typically replaced by an incrementally estimated standard deviation derived from observed traffic samples, since a fixed population parameter would require prior knowledge of the traffic distribution and periodic recalibration as baseline conditions evolve. This adaptive estimation approach is central to the proposed system design detailed in Section 3.4.

The two methods are complementary by design: CUSUM accumulates evidence of a persistent shift and is therefore better suited to slow-rate attacks, while EWMA smooths out noise and reacts more quickly to sudden volumetric changes. Sklavounos et al. confirmed that their combination on the NSL-KDD dataset reduces false positives while maintaining detection accuracy across multiple attack categories [53]. Christodoulou and Bi further demonstrated that a joint CUSUM-EWMA framework provides robust anomaly detection in time series traffic data, offering a practical balance between false positive rate and detection latency [54].

A fundamental assumption underlying both methods is that baseline traffic follows a Gaussian distribution when computing control limits. Real network traffic, however, frequently exhibits heavy-tailed behavior where large deviations occur far more often than a normal distribution would predict. Simmross-Wattenberg et al. addressed this by modeling traffic increments using alpha-stable distributions and constructing hypothesis tests based on this richer model. Their framework achieves substantially lower false positive rates on real traffic traces compared to Gaussian-assumption methods [55], highlighting that the statistical model must be matched to the actual distributional properties of the traffic being monitored.

### 2.3.3. Feature Selection for Lightweight Detection

Feature selection is the process of identifying the minimal subset of input variables that preserves the maximum discriminative information for a given detection task. In lightweight anomaly detection, it is a fundamental architectural requirement: every additional feature dimension translates directly into memory allocation, processing cycles, and detection latency. Kang and Kim [56] demonstrated that optimal feature subsets simultaneously reduce computational cost and improve classification accuracy by eliminating dimensions that introduce noise without contributing discriminative signal.

Feature selection methods fall into three families: filter methods evaluate features independently using statistical criteria such as mutual information or variance; wrapper methods measure subset impact on classifier performance; and embedded methods integrate selection into the learning process through regularization or splitting criteria. For resource-constrained deployment, filter methods are preferred since their independence from the classifier preserves the efficiency guarantees that motivate lightweight design. Zhang et al. [57] confirmed that a filter-then-wrapper pipeline yields near-optimal subsets at a fraction of the cost of pure wrapper approaches.

In SDN environments, the OpenFlow statistics API naturally constrains the observable feature space to a small set of first-order per-flow measurements, including packet counts, byte counts, flow counts, and duration. This constraint, rather than being a limitation, aligns directly with the requirements of lightweight detection: a bounded, computationally cheap feature space that can be evaluated in real time without external data collection [51]. Simmross-Wattenberg et al. [55] demonstrated that statistical inference over such features achieves detection accuracy comparable to deep learning while remaining orders of magnitude lighter. Table 2.2 summarizes the six flow-level features validated in the literature as the minimal sufficient set for characterizing anomalous traffic behavior in SDN environments [58][59][60][61].

Table 2.2 – Flow-level features for lightweight anomaly detection in SDN.

| **Feature** | **Definition** | **Behavioral Significance** |
|---|---|---|
| Packet Rate | delta_pkt / delta_t per source | Captures traffic volume anomalies |
| Byte Rate | delta_bytes / delta_t per source | Reflects bandwidth consumption patterns |

| | | |
|---|---|---|
| Flow Count | Active 5-tuples per window | Indicates connection diversity and rate |
| Unique Destinations | \|{dst_ip}\| per source/window | Measures destination spread behavior |
| Destination Entropy H(X) | -Sum p(xi) log2 p(xi) | Quantifies traffic distribution regularity |
| Dominant Ratio | max(dst_pkts) / total_pkts | Detects traffic concentration patterns |

### 2.3.4. Rule-Based and Threshold Detection

Rule-based detection operates by defining explicit conditions over observed traffic metrics, triggering an alert when one or more conditions are satisfied. Unlike learning-based approaches, rule-based systems require no training phase, no labeled dataset, and no model storage: their entire detection logic is encoded as a set of deterministic comparisons evaluated at each observation window. This makes them the most computationally predictable category of detection mechanisms, with a fixed and measurable overhead independent of traffic volume or pattern complexity.

The simplest form of rule-based detection is static thresholding, where a traffic metric is compared against a fixed boundary and a violation triggers an alert. While straightforward to implement, static thresholds suffer from a fundamental limitation in dynamic network environments: a boundary calibrated for one traffic regime becomes either too sensitive or too permissive as baseline conditions evolve. This fragility motivates the use of adaptive thresholding, where the decision boundary is continuously updated to reflect the current state of the network. Attar et al. [62] reviewed and enhanced adaptive dynamic threshold methods for anomaly detection in cyber-physical systems, demonstrating that dynamically adjusted boundaries consistently outperform static counterparts across varying traffic conditions. Chae et al. [63] further validated adaptive threshold mechanisms for anomaly-based intrusion detection, showing that statistically derived boundaries reduce false positive rates while maintaining detection sensitivity across evolving network traffic patterns.

Adaptive threshold mechanisms derive their decision boundaries from statistical estimators of normal traffic behavior, computed incrementally over observed flow windows rather than fixed offline baselines. The boundary is expressed as a function of the estimated mean and deviation of the monitored metric, with a sensitivity coefficient controlling the trade-off between false positive rate and detection responsiveness. This formulation yields a detection boundary that continuously tracks baseline traffic dynamics while remaining sensitive to abrupt anomalous deviations, without requiring any training phase or stored model.

### 2.3.5. Limitations of Lightweight Methods

Lightweight statistical methods offer significant advantages in terms of computational efficiency and deployment simplicity; however, the literature identifies several persistent limitations that

constrain their effectiveness in practice. Entropy-based approaches are particularly sensitive to threshold calibration, a boundary optimized for one traffic regime becomes unreliable as baseline conditions evolve, a problem documented by Santos-Neto et al. [64] who demonstrated that static entropy thresholds produce inconsistent detection rates across different traffic environments without dynamic adjustment mechanisms. Rule-based systems with fixed decision boundaries exhibit similar fragility, as confirmed by studies showing that statically derived thresholds generate elevated false positive rates under legitimate traffic fluctuations that resemble attack signatures [62][63].

A further limitation identified across the lightweight detection literature is the narrow scope of individual proposals. The majority of existing systems address either volumetric DDoS attacks or network scanning independently, without considering their co-occurrence or providing a unified detection framework capable of distinguishing between them [20][21]. Alhijawi et al. [65] noted in their comprehensive survey that most SDN detection solutions focus exclusively on DDoS, with scanning and reconnaissance attacks receiving comparatively little attention. Furthermore, Jia et al. [66] and Wang and Wang [67], despite proposing lightweight SDN-specific systems, do not provide mechanisms for automatically differentiating attack type prior to mitigation, a distinction that is operationally significant since DDoS and scanning require different response strategies. The absence of integrated, type-aware automated mitigation in most lightweight proposals represents a gap that the existing literature has not yet fully addressed.

Beyond scope limitations, lightweight statistical methods exhibit known failure modes that must be acknowledged. Entropy-based detection is vulnerable to evasion by slow-rate attackers who deliberately spread traffic across multiple destinations to maintain artificially high entropy values, effectively mimicking normal traffic distributions while still conducting reconnaissance [52]. EWMA-based estimators introduce a structural vulnerability during the warm-up period, where the smoothing mechanism averages out both normal variation and anomalous mutations simultaneously, leaving the baseline estimate unreliable before sufficient observations have been collected [49]. These known limitations directly informed the design choices of the proposed system, particularly the minimum observation threshold enforced before any classification decision is issued, as described in Section 3.4.1.

## 2.4 Related Work on Detection and Mitigation Systems in SDN

This section reviews existing complete system proposals for intrusion detection and mitigation in SDN environments, examining their architectural choices, detection strategies, attack coverage, and mitigation capabilities. Unlike the technique-level analysis presented in Sections 2.2 and 2.3, the focus here is on end-to-end systems evaluated within SDN contexts, with particular attention to their handling of controller overhead, attack type coverage, and automated response mechanisms.

### 2.4.1. DDoS Detection and Mitigation Systems

Ahuja and Singal [68] proposed a DDoS detection and prevention system for SDN leveraging OpenFlow statistics collected at the controller, using traffic rate monitoring and flow count

analysis to identify flooding behavior. Their system demonstrated effective detection under controlled Mininet simulations but relied on manually configured static thresholds, limiting adaptability to varying baseline traffic conditions. Rakissaga et al. [69] addressed the specific threat of Packet-In flooding attacks against SDN controllers, proposing a defense model that monitors Packet-In message rates and installs rate-limiting rules upon threshold violation. While their approach directly targets the controller saturation vector exploited by DDoS attacks, it operates on a single feature without considering the broader traffic context necessary to distinguish flooding from legitimate bursts of new flows.

Huang et al. [70] proposed FSDM, a fast recovery saturation attack detection and mitigation framework for SDN combining detection of control plane saturation with rapid flow rule deployment to restore normal operation. Their evaluation demonstrated significantly reduced recovery time compared to reactive-only approaches, highlighting the importance of mitigation speed as a performance metric independent of detection accuracy. Wang [71] proposed FFTC, a DDoS detection and defense mechanism based on flow feature analysis and threshold control in SDN, combining multiple flow-level features with a threshold-based classifier to identify attack traffic and install blocking rules via OpenFlow. Their system achieved low detection latency in simulation but did not address scanning attack behavior or provide mechanisms for distinguishing between attack types prior to mitigation.

Flauzac et al. [72] proposed an SDN architecture incorporating OpenFlow-based attack prevention, demonstrating that the programmability of the SDN control plane can be exploited to enforce security policies dynamically without external firewall infrastructure. Tang et al. [49] proposed FTMaster, a detection and mitigation system targeting low-rate flow table overflow attacks in SDN using AI-based classification to identify attack patterns that evade volume-based detectors. Their system is notable for addressing a stealthy attack variant that operates below conventional detection thresholds, demonstrating that flow table exhaustion can be achieved without generating the high packet rates typically associated with DDoS.

### 2.4.2. Scanning Detection Systems in SDN

Compared to DDoS detection, network scanning detection within controller-based SDN environments has received markedly less research attention, and the existing proposals exhibit significant limitations in scope and generalizability.

Shin et al. [73] proposed FRESCO, a modular security service framework for SDN that included scanning detection as one of several composable security modules. While FRESCO demonstrated the architectural feasibility of integrating scanning detection into the SDN control plane, its detection logic relied on predefined signatures and static rule matching rather than statistical flow analysis, limiting its ability to detect low-rate or distributed scanning campaigns that do not match known patterns.

Neu et al. [74] proposed a lightweight intrusion prevention system specifically targeting port scan detection in OpenFlow SDN networks, representing one of the few dedicated treatments of scanning in controller-based SDN. Their system monitored new flow rates and destination

diversity per source to identify scanning behavior and installed blocking rules upon detection. However, their approach addressed scanning in isolation without considering its co-occurrence with DDoS traffic, and their evaluation did not assess the impact of the detection mechanism on controller processing overhead under concurrent traffic loads.

Patil et al. [75] proposed a port scanning detection model targeting malicious TCP traffic in SDN using flow-level features to identify scanning patterns and mitigate their impact through flow rule modification. Their system demonstrated effective detection of TCP-based port scans but was limited to a single transport protocol, without generalizing to UDP or ICMP scanning variants. Critically, no mechanism was provided for differentiating scanning behavior from DDoS traffic when both are present simultaneously.

Miranda et al. [76] proposed p4SD, a lightweight port scan detection system for programmable networks based on P4 data plane programming. Their approach offloads scan detection entirely to the data plane, eliminating controller involvement and achieving detection latency at line rate. While p4SD represents a significant advance in detection efficiency, it requires P4-capable hardware and cannot be deployed in standard OpenFlow-based SDN environments.

Beyond SDN-specific proposals, established statistical paradigms for network scan detection exist in the broader literature and warrant acknowledgment. Threshold Random Walk (TRW), introduced by Jung et al., models connection attempts per source as a random walk between two stochastic processes representing benign and malicious behavior, requiring only four to five connection attempts to identify a scanner using Sequential Probability Ratio Test (SPRT) logic [77]. Alsaleh et al. subsequently demonstrated that TRW false positive rates grow proportionally with the transiency of network services, and that the algorithm is blind to vertical scanning patterns where a single source probes many ports on one destination [78]. Critically, both TRW and SPRT-based approaches rely on per-source connection state tracking maintaining records of successful and failed connection attempts at the switch or sensor level. In controller-based OpenFlow SDN, this per-source state maintenance conflicts with the stateless forwarding model of OpenFlow switches and imposes memory overhead on TCAM-constrained devices, making these methods architecturally incompatible with standard OpenFlow deployments without significant data plane modifications.

### 2.4.3. Hybrid and Flow-Based Analysis Systems

Giotis et al. [79] combined OpenFlow statistics with sFlow monitoring for entropy-based anomaly detection and mitigation in SDN, demonstrating that separating the data collection layer from the control plane reduces controller overhead while preserving detection effectiveness. Their work establishes an important architectural principle: detection performance and controller efficiency are not necessarily in conflict if the data collection strategy is designed carefully. Sayed et al. [80] proposed a comprehensive flow-based anomaly detection approach for DDoS in SDN integrating feature selection with deep learning, achieving strong detection performance on the InSDN dataset while demonstrating that feature dimensionality reduction directly impacts both accuracy and computational cost. Bhardwaj and Girdhar [81] analyzed network traffic behavior in SDN using the Ryu controller, providing empirical characterization of normal and anomalous

flow patterns that informs feature selection for detection systems operating within Ryu-based deployments.

## 2.5 Summary of Prominent Works

Table 2.3 summarizes the most prominent works reviewed in this chapter, organized by technique category with their key contributions and publication year.

Table 2.3 – Most Prominent Works in SDN Intrusion Detection.

| Technique | Reference | Key Contribution | Year |
|---|---|---|---|
| Classical ML | Buczak & Guven [40] | Foundational survey establishing RF, SVM, and DT as dominant paradigms for cyber security intrusion detection across multiple attack categories. | 2016 |
| Classical ML | Zhang et al. [41] | Comparative evaluation on KDD CUP99 and NSL-KDD confirming ensemble learning superiority with distinct per-algorithm trade-offs. | 2022 |
| Deep Learning | Bajri et al. [82] | CNN-LSTM hybrid architecture for real-time anomaly detection in SDN deployed on edge-enabled infrastructure, achieving competitive detection with reduced controller burden. | 2025 |
| Entropy-based | Wang et al. [70] | First entropy-based distributed DDoS detection mechanism in SDN using OpenFlow flow statistics with minimal controller overhead. | 2020 |
| Statistical SDN | Varalakshmi & Thenmozhi [83] | EDDA algorithm combining entropy fluctuation analysis with dynamic rate-limiting for SDN-IoT environments. | 2025 |
| Lightweight SDN | Jia et al. [65] | Lightweight DDoS detection scheme in SDN using flow statistics and adaptive thresholds with negligible controller overhead. | 2022 |
| Lightweight SDN | Wang & Wang [66] | SDN-Defend: lightweight online detection and mitigation combining statistical flow analysis with automated OpenFlow response. | 2022 |
| P4 data plane | Miranda et al [76] | Lightweight port scan detection offloaded entirely to the P4 data plane, achieving line-rate detection without any controller involvement. Requires P4-capable hardware | 2022 |

| | | | |
|---|---|---|---|
| Port Scan SDN | Neu et al. [74] | Lightweight IPS for port scan in OpenFlow SDN monitoring flow rate and destination diversity only dedicated scanning treatment in controller-based SDN. | 2018 |

## 2.6 Comparative Analysis

Table 2.4 synthesizes the reviewed systems across seven evaluation criteria: attack type covered, detection method, training requirement, computational overhead, real-time capability, automated mitigation, and SDN-specific design. The proposed system is included as the final row to position its contributions relative to the existing literature.

Table 2.4 – Comparative Analysis of SDN Intrusion Detection and Mitigation Systems.

| Reference | Attack Type | Method | Training | Overhead | Real-Time | Mitigation | SDN-Specific |
|---|---|---|---|---|---|---|---|
| Santos-Neto et al. [63] 2024 | DDoS | Entropy+ML hybrid | Yes | Moderate | Partial | No | Yes |
| Wang et al. [70] 2020 | DDoS | Shannon Entropy | No | Low | Yes | No | Yes |
| Ujjan et al. [71] 2021 | DDoS | Multi-feature Entropy | No | Low | Yes | No | Yes |
| Jia et al. [65] 2022 | DDoS | Lightweight stats | No | Low | Yes | Partial | Yes |
| Wang & Wang [66] 2022 | DDoS | Statistical+rules | No | Low | Yes | Yes | Yes |
| Ahuja & Singal [67] 2019 | DDoS | OpenFlow stats+rules | No | Low | Yes | Partial | Yes |
| Varalakshmi et al. [83] 2025 | DDoS | Entropy+stochastic | No | Low | Yes | Yes | Yes |
| Giotis et al. [79] 2014 | DDoS | OpenFlow+sFlow entropy | No | Low | Yes | Yes | Yes |
| Shin et al. [73] 2013 | Scanning +multiple | Signature+rules | No | Low | Yes | Partial | Yes |
| Neu et al. [74] 2018 | Scanning | Flow rate+dst diversity | No | Low | Yes | Partial | Yes |
| Patil et al. [75] 2021 | Scanning (TCP) | Flow features+rules | No | Low | Yes | Yes | Yes |
| Miranda et al [70] 2022 | Scanning | P4 data plane | No | Very low | Yes | Partial | No |
| Proposed System | DDoS+Scanning | Adaptive EWMA+rules | No | Low | Yes | Yes | Yes |

The table reveals three clear patterns. First, all ML and DL approaches require training, incur high overhead, and are unsuitable for real-time deployment within the SDN controller. Second, lightweight approaches that avoid training predominantly target DDoS only, with no unified system addressing both DDoS and scanning simultaneously. Third, among the few scanning-specific proposals, none provides automated type-aware mitigation integrated within an OpenFlow pipeline. The proposed system is the only entry satisfying all five operational

requirements simultaneously: no training, low overhead, real-time operation, automated mitigation, and unified DDoS-plus-scanning coverage.

## 2.7 Research Gap

The analysis presented in this chapter reveals three specific gaps that have not been addressed by any single prior proposal.

The first gap concerns unified attack coverage. Despite the co-existence of DDoS and scanning threats in real SDN environments, no existing lightweight system provides a training-free, real-time detection mechanism capable of jointly addressing both attack types within a single unified framework operating on the same feature set. Lightweight DDoS proposals do not account for scanning behavior, and the only scanning-specific proposals, Neu et al. [74] and Patil et al. [75], treat scanning in complete isolation from DDoS. Recent comprehensive surveys confirm that this unified treatment remains an open problem [65][54][55].

The second gap concerns type-aware automated mitigation. The majority of lightweight proposals either lack automated mitigation entirely or provide undifferentiated blocking that applies the same response regardless of attack type. The ability to differentiate attack type prior to response is operationally significant: DDoS and scanning attacks exhibit different traffic intensities, persistence patterns, and threat profiles, and may warrant different mitigation durations and response strategies. No existing lightweight system integrates this type-awareness into its mitigation pipeline.

The third gap concerns scanning detection in controller-based OpenFlow SDN. While P4-based frameworks [76] optimize data-plane detection, and statistical methods (TRW, SPRT) offer principled identification, they demand significant hardware modifications or switch-level state overhead [78]. Consequently, OpenFlow controllers remain highly vulnerable to scanning-induced Packet-In floods [68][84]. No recent proposal provides a complete detection and mitigation response operating purely within the controller using native flow statistics. The proposed system fills this gap by executing detection entirely at the controller level, requiring no switch-side state beyond standard OpenFlow 1.3 entries.

## 2.8 Conclusion

This chapter has provided a comprehensive survey of intrusion detection mechanisms for SDN environments across three analytical dimensions: data-driven approaches, lightweight statistical methods, and complete SDN-specific system proposals. Machine learning and deep learning approaches, while achieving high detection accuracy, introduce training dependencies, inference overhead, and dataset sensitivity that are incompatible with the real-time constraints of operational SDN controllers. Lightweight statistical methods address the overhead problem effectively but exhibit limitations in threshold calibration, adaptability to evolving baselines, and attack type coverage. SDN-specific system proposals, as synthesized in Tables 2.2 and 2.3, predominantly target DDoS in isolation, with scanning detection remaining a largely underexplored problem in the OpenFlow controller context. The comparative analysis confirms

that the existing literature has not produced a system simultaneously satisfying all operational requirements relevant to the SDN controller environment: absence of training, low computational overhead, real-time responsiveness, unified DDoS-and-scanning coverage, and integrated type-aware automated mitigation. This gap directly motivates the design objectives and architectural choices of the proposed system presented in Chapter 3.

Chapter

3

# A Lightweight Flow-Based Threat Detection and Mitigation in SDN

## 3.1 Introduction

This chapter presents the proposed lightweight flow-based detection and mitigation system designed to identify and block DDoS and scanning attacks in SDN environments while preserving controller performance. Based on the analysis of existing approaches in Chapter 2, this design adopts an adaptive statistical framework built on the Exponentially Weighted Moving Average as its core detection mechanism. The key property of this approach, which consists of continuously estimating normal per-source traffic behavior and deriving dynamic detection thresholds without any training phase, labeled dataset, or stored model, is central to the proposed system. While machine learning approaches offer stronger classification capabilities, they were excluded due to their training dependency and incompatibility with the resource constraints of the SDN control plane. The proposed system combines six complementary flow-level features with adaptive thresholding and formally defined classification conditions to achieve type-aware detection, coupled with an automated two-table OpenFlow mitigation pipeline that installs DROP rules upon confirmed attack detection with timeout-based automatic unblocking once the threat subsides.

## 3.2 Overview of the Proposed System

The proposed system is built on the SDN centralized control paradigm, extended with a lightweight adaptive statistical detection mechanism to identify and block DDoS and scanning attacks directly within the controller without degrading its primary forwarding responsibilities. It allows the OS-Ken controller to continuously observe per-source traffic behavior through aggregated OpenFlow flow statistics, classify each source as normal, DDoS, or scanning using adaptive thresholds derived from live baseline estimates, and automatically install blocking rules in the data plane upon confirmed attack detection, all within a single polling cycle and without any prior training phase or external processing infrastructure. The two-table OpenFlow pipeline ensures that mitigation rules installed in Table 0 intercept attack traffic before it reaches the forwarding logic in Table 1, keeping the entire detection-to-blocking path within the SDN control plane.

The overall system workflow is structured as follows:

1. The OS-Ken controller initializes the two-table pipeline on each connected switch, installing a default goto-table rule in Table 0 and a default Packet-In rule in Table 1.
2. At every one-second polling interval, the controller issues an OFPFlowStatsRequest to all connected switches and collects per-flow packet and byte counters.
3. For each active source IP, the controller computes six flow-level features from the collected statistics: packet rate, byte rate, flow count, unique destination count, destination entropy, and dominant traffic ratio.
4. Each feature is evaluated against a per-source adaptive threshold derived from an EWMA baseline estimator, continuously updated to reflect the source's current normal behavior.
5. The detection engine applies formally defined boolean classification conditions to determine whether the source exhibits DDoS behavior, scanning behavior, or normal traffic patterns.
6. Upon confirmed attack detection, the mitigation engine issues an OFPFlowMod message to install a priority DROP rule in Table 0, blocking all further traffic from the identified source for 60 seconds with automatic expiry.

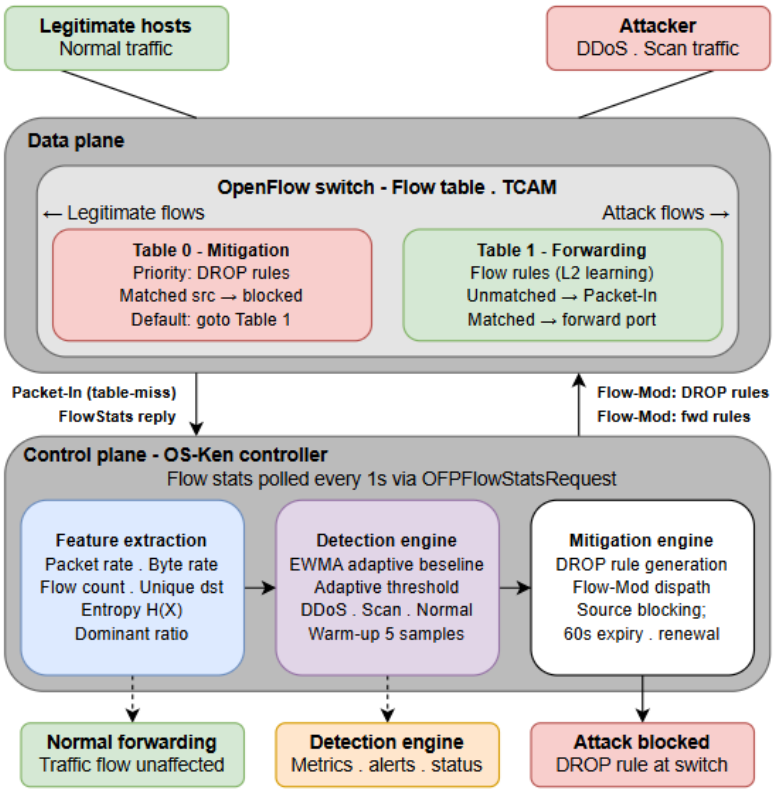


Figure 3.1 – High-level overview of the proposed adaptive detection system.

## 3.3 Flow-Level Feature Extraction

### 3.3.1. OpenFlow Statistics Collection Mechanism

The system relies exclusively on flow-level statistics natively exposed by the OpenFlow 1.3 protocol, requiring no packet inspection and no external monitoring infrastructure. At each one-second polling cycle, the controller dispatches an OFPFlowStatsRequest to every connected switch. Each switch responds with per-flow entries carrying match fields and cumulative packet and byte counters. Only forwarding-table entries (Table 1, priority 1) are processed; mitigation DROP rules and default table-miss rules are excluded to prevent feedback between detection and mitigation layers. Algorithm 3.1 formalizes the collection and accumulation procedure. The one-second polling interval was selected to balance detection responsiveness with controller processing overhead in the evaluated topology. Extending this to an adaptive polling model that dynamically adjusts sampling frequency based on active flow count represents a natural evolution of the design for larger-scale deployments.

```
Algorithm 3.1: Flow Statistics Collection and Per-Source Accumulation
Input  : Datapaths D, previous counter state prev_stats
Output: Per-source accumulators for all active sources
 1:  for each datapath dp in D do
 2:      send OFPFlowStatsRequest(dp, table_id=ALL)
 3:  end for
 4:  upon receiving OFPFlowStatsReply(body, dp) do
 5:      for each flow entry e in body do
 6:          if e.priority != 1 or e.table_id != FORWARDING_TABLE
then
 7:              continue
 8:          end if
 9:          src, dst_target <- extract_identity(e.match)
               // dst_target = (dst_ip, proto, port)
10:          if src = null or dst_target = null then continue end if
11:          key <- (dp.id, src, dst_target)
12:          pkt_delta  <- max(0, e.packet_count -
prev_stats[key].packets)
13:          byte_delta <- max(0, e.byte_count  -
prev_stats[key].bytes)
14:          prev_stats[key] <- (e.packet_count, e.byte_count)
15:          if pkt_delta = 0 then continue end if
16:          src_pkt_delta[src]              += pkt_delta
17:          src_byte_delta[src]             += byte_delta
18:          src_targets[src].add(dst_target)
19:          src_flow_count[src]             += 1
20:          src_target_pkt[src][dst_target] += pkt_delta
21:      end for
22:      compute feature vectors from accumulators (Section 3.3.2)
23: end upon
```

### 3.3.2. Feature Definition and Behavioral Significance

From the accumulated per-source statistics, the system derives six scalar features per source per polling cycle. Let S denote the set of unique destination targets observed for source src within the current window, where each target is a triple (dst_ip, proto, port), making S sensitive to both destination IP diversity in horizontal scanning and port diversity in vertical scanning. Let delta_pkt(t) and delta_bytes(t) denote the packet and byte deltas directed toward target t in S.

**Packet rate:** total packets from src across all active flows in the window.

$$f_1 = \sum_{t \in S} \Delta pkt(t) \tag{3.1}$$

**Byte rate:** total bytes from src across all active flows in the window.

$$f_2 = \sum_{t \in S} \Delta bytes(t) \tag{3.2}$$

**Flow count:** number of distinct active 5-tuples with non-zero packet delta.

$$f_3 = \left|\{(src, dst_{ip}, proto, port) : \Delta pkt > 0\}\right| \quad (3.3)$$

**Unique destination count:** number of distinct destination targets contacted.

$$f_4 = |S| \quad (3.4)$$

**Destination entropy:** Shannon entropy over the per-target traffic distribution.

$$f_5 = -\sum_{t \in S} p(t) \log_2 p(t) \;\; , \;\; p(t) = \frac{\Delta pkt(t)}{\sum_{t' \in S} \Delta pkt(t')} \quad (3.5)$$

**Dominant traffic ratio:** fraction of total traffic directed toward the single most contacted target.

$$f_6 = \frac{max_{t \in S} \Delta pkt(t)}{\sum_{t' \in S} \Delta pkt(t')} \quad (3.6)$$

Note that $f_5$ and $f_6$ are complementary by construction: $f_6$ is high precisely when $f_5$ is low, and vice versa. This inverse relationship is the statistical mechanism that separates DDoS from scanning behavior in the classification conditions of Section 3.5. Table 3.1 summarizes the behavioral significance of each feature across the three traffic classes.

Table 3.1 – Feature Behavioral Divergence Across Traffic Classes.

| Feature | Normal traffic | DDoS | Scanning |
|---|---|---|---|
| Packet rate f1 | Moderate, variable | Very high, sustained | Moderate to low |
| Byte rate f2 | Moderate | Very high | Low to moderate |
| Flow count f3 | Low to moderate | Very low: few flows, high volume | Very high: many distinct flows |
| Unique destination f4 | Low to moderate | Very low: single target | Very high: many target or ports |
| Dest entropy f5 | Moderate to high | Very low: traffic concentrated | Very high: traffic dispersed |
| Dominant ratio f6 | Moderate | Very high >=0.6 | Very low <=0.5 |

## 3.4 Adaptive Baseline Estimation Using EWMA

### 3.4.1. Per-Source State Initialization and Warm-Up

The detection engine maintains a dedicated statistical state for each source IP address, initialized upon its first appearance and updated across polling cycles. For each source src, the system stores the state vector:

$$\Omega_{src} = \{\mu_f, \sigma_f, n\}, \forall f \in \{f_1, f_2, f_3, f_4, f_5, f_6\} \quad (3.7)$$

where $\mu_f$ is the EWMA mean estimate for feature f, $\sigma_f$ is the EWMA deviation estimate, and n is the observation count. All estimates initialize to zero: $\mu_f(0) = 0$, $\sigma_f(0) = 0$, n = 0. Note that $f_6$ does not maintain an EWMA state as it is used directly in classification conditions but not for threshold computation.

A mandatory warm-up period is enforced before any classification decision is issued. The system withholds attack classification for a source until it has accumulated a minimum of $n_{min} = 5$ observation samples, ensuring that the baseline estimates have converged sufficiently to produce meaningful thresholds. EWMA updates proceed normally during warm-up, but the classification conditions of Section 3.5 are not evaluated.

### 3.4.2. Threshold Derivation and Update Rule

At each polling cycle, the system updates per-source EWMA estimates and derives adaptive detection thresholds. The update proceeds in two steps.

Cold start. If $\mu_f = 0$, indicating the first non-empty observation for this source, the mean is seeded from the observed value and the deviation is initialized to a minimum floor:

$$if\, \mu_f = 0: \quad \mu_f \leftarrow x_f\,, \qquad \sigma_f \leftarrow 1.0 \tag{3.8}$$

EWMA update. For all subsequent observations, with smoothing factor $\alpha = 0.2$:

$$\mu_f(n) = \alpha x_f(n) + (1-\alpha)\mu_f(n-1) \tag{3.9}$$

$$\sigma_f(n) = \max(1.0, \alpha|x_f(n) - \mu_f(n)| + (1-\alpha)\sigma_f(n-1)) \tag{3.10}$$

The deviation in Equation 3.10 is computed against the newly updated mean $\mu_f$(n), making it a mean absolute deviation tracker. The floor of 1.0 prevents threshold collapse during abnormally stable traffic periods.

Threshold derivation. The adaptive detection threshold for each feature is:

$$\theta_f(n) = \max(\theta_{f,min}, \mu_f(n) + k\sigma_f(n)) \tag{3.11}$$

where k = 1.5 is the sensitivity multiplier and $\theta_{f,min}$ are feature-specific minimum floors: $\theta_{pkt,min} = 5$, $\theta_{byte,min} = 400$, $\theta_{dst,min} = 2$, $\theta_{flow,min} = 2$, $\theta_{entropy,min} = 1.2$. For the entropy feature, the multiplier is halved to 0.5 * k = 0.75, reflecting the higher natural variance of entropy relative to rate-based features:

$$\theta_{entropy}(n) = \max(1.2, \mu_{entropy}(n) + 0.75\sigma_{entropy}(n)) \tag{3.12}$$

A feature anomaly flag and composite anomaly score are derived as:

$$\alpha_f(n) = 1\left[x_f(n) > \theta_f(n)\right] \tag{3.13}$$

$$A(n) = \sum_{f \in F} a_f(n) \in \{0,1,2,3,4,5\} \tag{3.14}$$

**Algorithm 3.2:** Per-Source EWMA Baseline Update and Threshold Derivation

```
Input : Feature vector x = (x_f1,...,x_f5), source state Omega_src =
{mu_f, sigma_f, n}
Output: Thresholds {theta_f}, anomaly flags {a_f}, anomaly score A
Parameters: alpha = 0.2,  k = 1.5,  n_min = 5
            theta_min = {pkt:5, byte:400, dst:2, flow:2, entropy:1.2}

 1:  for each feature f in {f1, f2, f3, f4, f5} do
 2:      if mu_f = 0 then
 3:          mu_f <- x_f  ;  sigma_f <- 1.0                  // Eq. 3.8
 4:      else
 5:          mu_f <- alpha * x_f + (1-alpha) * mu_f          // Eq. 3.9
 6:          sigma_f <- max(1.0, alpha*|x_f-mu_f| + (1-
alpha)*sigma_f) // Eq. 3.10
 7:      end if
 8:      if f = entropy then
 9:          theta_f <- max(theta_min_f, mu_f + 0.5*k*sigma_f)   //
Eq. 3.12
10:      else
11:          theta_f <- max(theta_min_f, mu_f + k*sigma_f)        //
Eq. 3.11
12:      end if
13:      a_f <- 1 if x_f > theta_f else 0                         //
Eq. 3.13
14:  end for
15:  n <- n + 1
16:  A <- Sum a_f                                                  //
Eq. 3.14
17:  if n < n_min then return WARM-UP end if
18:  return {theta_f}, {a_f}, A
```

The current implementation polls all connected switches unconditionally at every one-second interval. While this design is sufficient for the evaluated topology, it introduces a control-channel bottleneck in larger deployments where the volume of flow statistics responses may introduce counter-arrival jitter that skews EWMA baseline estimates. A more scalable design would adopt an adaptive polling strategy, increasing sampling frequency when anomaly scores rise above a baseline threshold and reducing it during stable periods, or transitioning to an asynchronous trigger-based model where switches report statistics only upon detecting predefined flow count thresholds. Such an extension is identified as a direction for future work in the General Conclusion.

### 3.4.3. Justification for EWMA as the Baseline Estimator

The choice of EWMA was made on the basis of four simultaneous requirements: no training, low overhead, continuous adaptability, and O(1) incremental update per feature per cycle. Table 3.2 positions EWMA against principal alternatives. EWMA is the only method satisfying all four requirements simultaneously, as validated by Ye et al. [49], Sklavounos et al. [53], and Christodoulou and Bi [54]. Static thresholds fail to adapt to evolving baselines, producing elevated false positive rates as confirmed by Attar et al. [60] and Chae et al. [61]. CUSUM detects

persistent shifts but does not maintain a continuously updated mean estimate from which adaptive thresholds can be derived [53][54]. ML-based estimators adapt to baseline changes but require training phases and incur inference overhead incompatible with real-time controller operation [20][21].

Table 3.2 – Comparison of Baseline Estimation Methods for SDN Anomaly Detection.

| **Method** | **Training** | **Overhead** | **Adaptive baseline** | **Incremental update** |
|---|---|---|---|---|
| Static threshold | No | Very low | No | N/A |
| CUSUM | No | Low | Partial | Yes |
| EWMA (proposed) | No | Low | Yes | Yes |
| ML-based estimator | Yes | High | Yes | No |

## 3.5 Attack Classification Logic

The detection engine evaluates each source against two sets of formally defined conditions derived from the behavioral divergence established in Table 3.1. Classification proceeds only after the warm-up period has elapsed (n >= 5). Sources satisfying the DDoS profile take priority over the scanning profile; a source cannot be classified as both simultaneously within the same cycle.

### 3.5.1. DDoS Detection Conditions

A source src is classified as DDoS if and only if all six conditions in Table 3.3 hold simultaneously. The conjunction of all six conditions is required to minimize false positives. The byte rate condition uses 80% of the adaptive threshold to account for packet size variability in flooding attacks.

Table 3.3 – DDoS Classification Conditions.

| **Cond.** | **Expression** | **Behavioral meaning** |
|---|---|---|
| C1 | f1 >= max(theta_pkt, 15) | Elevated packet rate above adaptive threshold |
| C2 | f2 >= max(0.8 * theta_byte, 400) | Elevated byte rate (80% of threshold, accounts for packet size variance) |
| C3 | f4 <= 2 | Very low destination diversity: single target |
| C4 | f3 <= 3 | Very low flow count: high volume concentrated on few destinations |
| C5 | f5 <= 1.0 | Very low entropy indicating highly concentrated traffic |
| C6 | f6 >= 0.6 | Dominant ratio above 60%, one target receives most traffic |

The formal boolean predicate is:

$$DDoS(src) = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6 \quad (3.15)$$

The conjunction of all six conditions is deliberately strict to minimize false positives. The six-condition predicate was validated against the experimental scenarios of Chapter 4, where all six conditions were consistently satisfied by attacking sources while never co-occurring in legitimate traffic. However, real-world DDoS variants including low-rate flooding, rate-limited attacks, and multi-source distributed campaigns may individually satisfy fewer than all six conditions simultaneously, representing a known operational boundary of this design. Addressing such scenarios through adaptive conjunction relaxation or cross-source aggregation is identified as a direction for future work.

### 3.5.2. Scanning Detection Conditions

A source src is classified as scanning if DDoS(src) is false and all five conditions in Table 3.4 hold simultaneously. Condition C5 explicitly upper-bounds the packet rate, separating scanning from volumetric flooding. The entropy condition uses 80% of the adaptive threshold to accommodate slow or distributed scanning campaigns.

Table 3.4 – Scanning Classification Conditions.

| Cond. | Expression | Behavioral meaning |
|---|---|---|
| C1 | f4 >= max(3, ceil(theta_dst)) | High destination diversity above adaptive threshold |
| C2 | f3 >= max(3, ceil(theta_flow)) | High flow count above adaptive threshold |
| C3 | f5 >= max(1.3, 0.8*theta_entropy) | High entropy indicating dispersed traffic (relaxed to 80% of threshold) |
| C4 | f6 <= 0.5 | Low dominant ratio, no single dominant destination |
| C5 | f1 <= max(1.5*theta_pkt, 25) | Moderate packet rate, distinguishing scanning from volumetric flooding |

The formal boolean predicate is:

$$Scan(src) = \neg DDoS(src) \wedge C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \quad (3.16)$$

### 3.5.3. Anomaly Scoring and Alert Emission

For sources satisfying neither DDoS nor scanning profile, the composite anomaly score A (Equation 3.14) provides a secondary signal. Sources with A >= 2 are logged as general anomalies without mitigation being applied. Alerts are governed by a cooldown mechanism: for each (source, label) pair, at most one alert is emitted per five-second window, suppressing redundant notifications during sustained attacks. Algorithm 3.3 formalizes the complete classification and alert emission procedure.

```
Algorithm 3.3: Attack Classification and Alert Emission
Input  : Feature vector x, thresholds {theta_f}, anomaly flags {a_f},
score A,
        source state Omega_src, last_alert_time, datapath dp
Output: Classification label in {NORMAL, DDOS, SCAN, ANOMALY}

1:  if n < n_min then return NORMAL end if
2:  ddos <- (f1 >= max(theta_pkt,15)) AND (f2 >=
max(0.8*theta_byte,400))
              AND (f4 <= 2) AND (f3 <= 3) AND (f5 <= 1.0) AND (f6 >=
0.6)
3:  scan <- (f4 >= max(3,ceil(theta_dst))) AND (f3 >=
max(3,ceil(theta_flow)))
              AND (f5 >= max(1.3, 0.8*theta_entropy)) AND (f6 <= 0.5)
              AND (f1 <= max(1.5*theta_pkt, 25))
 4:  if ddos then
 5:      emit_alert('DDOS', src)                    // cooldown = 5 s
 6:      apply_mitigation(dp, src, 'DDOS')
 7:      return DDOS
 8:  else if scan then
 9:      emit_alert('SCAN', src)
10:      apply_mitigation(dp, src, 'SCAN')
11:      return SCAN
12:  else if A >= 2 then
13:      emit_alert('ANOMALY', src)        // log only, no mitigation
14:      return ANOMALY
15:  end if
16:  return NORMAL
```

## 3.6 The Two-Table OpenFlow Mitigation Pipeline

The mitigation pipeline operates entirely within the OpenFlow flow table infrastructure, requiring no external blocking devices. It separates traffic interception from forwarding logic through a two-table architecture in which all traffic enters Table 0 (mitigation table) first, before reaching Table 1 (forwarding table).

### 3.6.1 Pipeline Architecture and Flow Entry Design

Upon connection of a new switch, the controller clears all existing flow rules and initializes the two-table pipeline as follows:

- All existing flow rules are deleted from all tables via OFPFlowMod with command OFPFC_DELETE and table_id=OFPTT_ALL.
- A default goto-table rule is installed in Table 0: match=*, priority=0, instruction=GotoTable(1). This ensures unblocked traffic passes through to Table 1.
- A default table-miss rule is installed in Table 1: match=*, priority=0, action=OUTPUT(CONTROLLER). This triggers Packet-In for any flow with no matching forwarding rule.

The architectural invariant established by this initialization is: all traffic enters Table 0 first. A source with an active DROP rule is intercepted and discarded in Table 0 before reaching Table 1. A source without a DROP rule passes through Table 0 via the default goto instruction and reaches Table 1, where normal L2 forwarding applies. Figure 3.2 illustrates the complete pipeline structure including match-action entries and Flow-Mod installation paths.

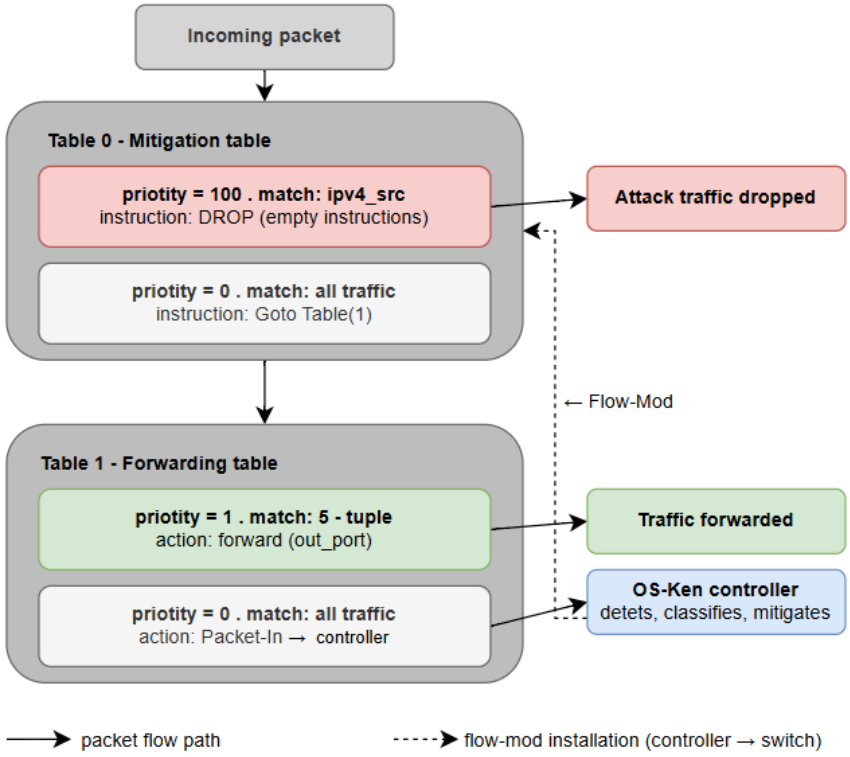


Figure 3.2 – Two-table OpenFlow pipeline showing packet processing sequence and Flow-Mod installation paths.

**Algorithm 3.4:** Two-Table Pipeline Initialization

**Input :** datapath dp

```
1:  send OFPFlowMod(dp, table_id=ALL, command=DELETE,
        out_port=ANY, out_group=ANY, match=*)      // clear all tables
2:  send OFPFlowMod(dp, table_id=0, priority=0,
        match=*, instruction=GotoTable(1))        // Table 0 default
3:  send OFPFlowMod(dp, table_id=1, priority=0,
        match=*, action=OUTPUT(CONTROLLER))      // Table 1 table-miss
```

### 3.6.2 DROP Rule Installation, Expiry, and Renewal

Upon confirmed attack detection, the mitigation engine installs a DROP rule in Table 0 for the attacking source IP. The rule is specified as:

$$Rule = \left\{ \begin{matrix} table_{id} = 0, priority = 100, match = \{ipv4_{src}: src_{ip}\}, \\ instructions = [\,], hard_{timeout} = 60 \end{matrix} \right\} \quad (3.17)$$

An empty instruction set causes the switch to silently discard all packets from the matched source without generating Packet-In events, removing the attacker from the controller's processing loop entirely. The priority of 100 ensures the DROP rule takes precedence over the default goto rule at priority 0.

A renewal mechanism prevents rule expiry during ongoing attacks. Before installing a rule, the system evaluates:

$$if\,(now - last_install) < (T_{timeout} - T_{renew}): skip \tag{3.18}$$

$$T_{timeout} = 60s\,,\quad T_{renew} = 10s$$

A rule is reinstalled only if it is within the last 10 seconds of its 60-second lifetime, ensuring continuous blocking without unnecessary Flow-Mod overhead during stable attack periods. Rules that are not renewed expire automatically via the OpenFlow hard_timeout mechanism, unblocking the source without controller intervention. Algorithm 3.5 formalizes the complete mitigation procedure.

**Algorithm 3.5:** DROP Rule Installation with Renewal Logic

```
Input : datapath dp, src_id, attack_type

 1:  (src_type, src_value) <- parse_src_id(src_id)
 2:  now <- current_time()
 3:  last_install <- mitigated_sources.get((dp.id, src_id), 0)
 4:  if (now - last_install) < (60 - 10) then
 5:      return
// rule still active, skip
 6:  end if
 7:  match <- build_match(src_type, src_value)// ipv4_src or ipv6_src
 8:  send OFPFlowMod(dp, table_id=0, priority=100,
          hard_timeout=60, match=match, instructions=[])
 9:  mitigated_sources[(dp.id, src_id)] <- now
10:  update Prometheus metrics (mitigated, expiry, total,
active_count)
```

## 3.7 Computational Complexity Analysis

Let F denote the number of active flow entries across all connected switches, S the number of unique source IPs (S <= F), and B the number of currently blocked sources (B <= S). Per polling cycle, the system performs the following operations:

Flow entry processing: O(F), covering one delta computation and source accumulation per flow entry.

EWMA update: O(S), covering five mean-deviation update pairs per source each requiring O(1) operations.

Classification: O(S), involving a constant number of boolean comparisons per source.

Mitigation: O(B), with at most one OFPFlowMod per currently blocked source per renewal window.

Total time complexity per polling cycle: O(F), since F >= S >= B always holds. Total space complexity: O(F + S) for the counter state and per-source statistical state. Table 3.5 positions the proposed system against ML-based approaches, confirming the lightweight claim established in Chapter 2.

Table 3.5 – Computational Complexity Comparison per Detection Cycle.

| **Approach** | **Time per cycle** | **Space** | **Training** |
|---|---|---|---|
| Static threshold | O(F) | O(1) | None |
| EWMA+rules (proposed) | O(F) | O(F+S) | None |
| Random Forest inference | O(F*D*T) | O(D*T+model) | Required |
| LSTM inference | $O(F*D*H^2)$ | $O(H^2+model)$ | Required |

D denotes feature dimensions, T the number of trees in Random Forest, and H the LSTM hidden state dimension. Both ML methods require O(F) feature extraction before inference, making their total per-cycle cost strictly greater than O(F). The proposed system achieves O(F) without any inference phase, stored model parameters, or periodic retraining overhead.

In practice, for a typical SDN environment with F <= 10,000 active flow entries and S <= 500 concurrent sources, the O(F) operations per polling cycle complete in well under one millisecond on commodity hardware, a negligible fraction of the one-second polling interval. This leaves the controller's processing capacity entirely available for its primary forwarding function throughout the detection cycle, confirming that the proposed system introduces no measurable latency penalty under normal operating conditions and remains effective as network scale increases.

## 3.8 Conclusion

This chapter has presented the complete design and implementation of the proposed lightweight flow-based detection and mitigation system for SDN. The system directly addresses the three gaps identified in Chapter 2. The first gap, the absence of a unified lightweight system covering both DDoS and scanning simultaneously, is addressed through formally defined classification conditions operating on the same six-dimensional feature space and distinguishing between both attack types without any training phase. The second gap, the lack of type-aware automated mitigation, is addressed through the two-table OpenFlow mitigation pipeline that installs differentiated DROP rules upon confirmed detection and revokes them automatically upon hard_timeout expiry. The third gap, the underexplored treatment of scanning-induced controller exhaustion in controller-based OpenFlow SDN, is addressed through flow-level scanning detection operating natively within the OS-Ken controller without requiring any data-plane programmability beyond standard OpenFlow 1.3 support. The computational complexity of O(F)

per polling cycle confirms the system's suitability for deployment within the resource constraints of the SDN control plane. The experimental evaluation is presented in Chapter 4.

Chapter

4

# Experimental Evaluation and Results

## 4.1 Introduction

This chapter presents the experimental setup, evaluation methodology, and results of the proposed lightweight flow-based detection and mitigation system. All experiments were conducted in a controlled Mininet-based SDN testbed with synthetically generated traffic, ensuring complete ground truth knowledge across all scenarios. The evaluation covers two dimensions: detection performance and operational efficiency. The chapter concludes with a discussion of system behavior, strengths, and operational boundaries.

## 4.2 Simulation Environment

### 4.2.1. Hardware and Virtualization Platform

All experiments were conducted on a VMware Workstation virtual machine running Ubuntu 24.04.4 LTS, allocated 8.6 GB of RAM, 2 virtual processors, and 50 GB of storage. The host machine is equipped with an Intel Core i7-8665U CPU at 1.90 GHz. The NAT network adapter configuration isolates the testbed from external network interference, ensuring controlled and reproducible experimental conditions.

### 4.2.2. Software Stack and Tools

The experimental environment relies on a set of open-source tools covering network emulation, SDN control, monitoring, and traffic generation. Table 4.1 summarizes the complete software stack with version information.

Table 4.1 – Experimental Software Stack.

| Component | Tool | Version | Role |
|---|---|---|---|
| Operating System | Ubuntu LTS | 24.04.4 | Virtualized experiment host |
| Network emulator | Mininet | 2.3.0 | SDN testbed emulation |
| SDN controller | OS-Ken | 2.8.1 | OpenFlow 1.3 control plan |
| Data plane | Open vSwitch | 3.3.4 | OpenFlow 1.3 switch emulation |
| Metrics collection | Prometheus | 2.48.1 | Real-time metrics scraping |
| Visualization | Grafana | 10.4.2 | Metrics dashboard and monitoring |
| DDoS generation | Hping3 | 3.0.0-α-2 | Flood traffic generation |
| Scan generation | Nmap | 7.94SVN | Port and host generation |
| Normal traffic | Iperf | 2.1.9 | Legitimate TCP/UDP traffic generation |
| Language | Python | 3.12.3 | Controller application runtime |

### 4.2.3. Network Topology

The evaluation topology is a two-level tree network instantiated using the Mininet tree topology constructor with depth 2 and fanout 3, producing one root switch, three edge switches, and nine hosts distributed evenly across the edge layer. All switches operate in OpenFlow 1.3 secure mode with a remote controller configuration, connecting to the OS-Ken controller instance running on the same host at 127.0.0.1. The topology is instantiated using the following command:

```
sudo mn --topo tree,2,3 --switch ovs,protocols=OpenFlow13,
        failMode=secure --controller=remote,ip=127.0.0.1
```

This configuration was chosen to ensure reproducibility and complete ground truth knowledge across all scenarios. The scalability of the detection overhead is bounded by this topology scale:

the per-cycle complexity is O(D x F), where D is the number of connected datapaths and F the average active flow entries per switch. For the evaluated configuration of D=4 datapaths and nine hosts, this remains negligible. In larger deployments, polling overhead grows proportionally and should be re-evaluated accordingly. Extending the evaluation to larger standardized topologies such as Fat-Tree and replaying real-world packet traces is identified as a direction for future work.

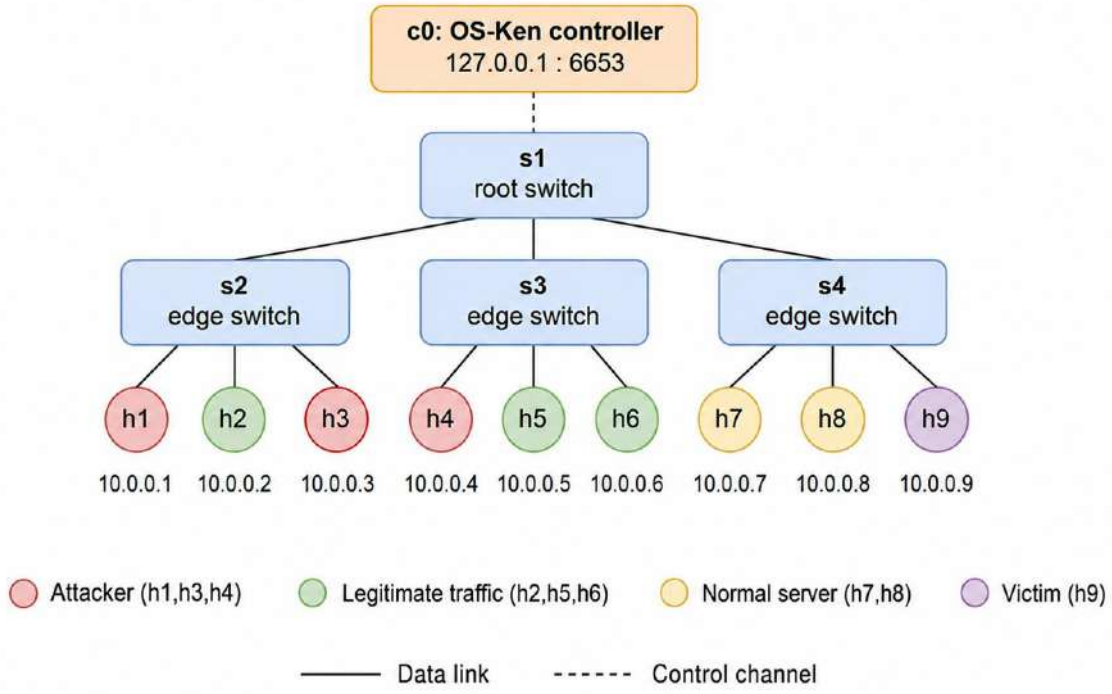


Figure 4.1 – Tree (2,3) evaluation topology with host role assignments.

## 4.3 Traffic Generation Methodology

All traffic was synthetically generated within the Mininet testbed. No external datasets were used, ensuring complete ground truth knowledge at every point in the evaluation timeline. Three traffic categories were generated: normal traffic representing legitimate network activity, DDoS traffic simulating volumetric flooding attacks, and scanning traffic simulating network reconnaissance behavior.

### 4.3.1. Normal Traffic Generation

Normal traffic was generated using three complementary tools to simulate realistic and diverse legitimate network activity. ICMP connectivity was established through standard ping exchanges between host pairs. TCP throughput traffic was generated using iperf between designated client and server hosts, simulating sustained legitimate data transfers. HTTP traffic was simulated by serving files via Python's built-in HTTP server on port 80 and retrieving them using wget, producing short-lived TCP flows typical of web browsing behavior.

### 4.3.2. DDoS Traffic Generation

DDoS traffic was generated using hping3 and ping, simulating three flooding variants targeting the victim host h9 (10.0.0.9) from multiple simultaneous attackers h1, h3, and h4.

ICMP flood was generated using the Linux ping flood option directed at the victim from multiple attackers simultaneously, producing high-rate ICMP echo request streams. TCP SYN flood was generated using hping3 with the SYN flag enabled at fast and flood rates targeting port 80. The flood mode removes inter-packet delays, generating the maximum possible packet rate from each attacking host. UDP flood was generated using iperf in UDP mode at 50 Mbps per attacking host, producing aggregate traffic sufficient to saturate the victim and generate sustained table-miss events.

### 4.3.3. Scanning Traffic Generation

Scanning traffic was generated using nmap and netcat, simulating both horizontal and vertical scanning behaviors originating from h1 targeting multiple hosts and ports across the network. Host discovery scanning was performed using nmap ping sweep mode across the 10.0.0.0/24 subnet. Port scanning was performed using nmap SYN scan mode, fast scan mode, and service version detection, each generating large numbers of distinct destination port flows. Multi-target netcat probes were issued in parallel toward multiple hosts on varying ports, simulating distributed reconnaissance that produces high destination entropy and flow count without elevated packet rates.

## 4.4 Attack Scenarios

Three controlled attack scenarios were designed to evaluate the detection and mitigation capabilities of the proposed system. Each scenario was executed independently with legitimate background traffic running concurrently to assess detection accuracy under mixed traffic conditions.

Table 4.2 – Attack Scenarios.

| Scenario | Attack Type | Attacker(s) | Target | Tool | Traffic Profile |
|---|---|---|---|---|---|
| S1 | ICMP Flood | h1, h3, h4 | h9 | Ping -f | High-rate ICMP echo requests from 3 simultaneous sources toward single victim |
| S2 | TCP/UDP Flood | h1, h3, h4 | h9 | hping3 --flood, ping -f | TCP SYN flood and ICMP flood at maximum rate from three simultaneous sources toward single victim |
| S3 | Network scan | h1 | h2-h9 | nmap -sS, nmap -F, nc -z | SYN scan, fast scan, and multi-target port probing across subnet |

Scenarios S1 and S2 were designed to trigger the DDoS classification conditions of Section 3.5.1 by producing high packet rates, low destination diversity, and elevated dominant ratio. Scenario S3 was designed to trigger the scanning classification conditions of Section 3.5.2 by producing high destination count, high entropy, and low dominant ratio without elevated packet volume. Legitimate traffic was generated concurrently from h3, h5, h6, h7, and h8 throughout all scenarios to validate that the detection mechanism correctly classifies benign sources as normal.

## 4.5 Evaluation Metrics

The proposed system is evaluated across two complementary dimensions: detection performance and operational efficiency. All metrics are derived from controller logs and Prometheus time-series data collected during each experimental scenario.

### 4.5.1. Detection Performance Metrics

Since the proposed system produces deterministic binary classifications per observation window, detection performance is evaluated through a confusion matrix constructed by comparing system output labels against ground-truth labels derived from the controlled traffic generation timeline. Four base quantities are defined for each scenario:

- **TP:** observation window during an active attack correctly classified as DDOS or SCAN
- **TN:** observation window during normal traffic correctly classified as NORMAL
- **FP:** observation window during normal traffic incorrectly classified as DDOS or SCAN
- **FN:** observation window during an active attack classified as NORMAL

From these four quantities, the following metrics are derived:

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4.3}$$

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{4.4}$$

### 4.5.2. Operational Efficiency Metrics

Operational efficiency metrics quantify the computational overhead introduced by the detection mechanism on the OS-Ken controller. All metrics are derived directly from controller log timestamps and Prometheus scrape data, requiring no additional instrumentation beyond what the system already exports. Table 4.3 summarizes the five operational metrics and their measurement sources.

Table 4.3 – Operational Efficiency Metrics.

| Metric | Definition | Source |
|---|---|---|
| Detection latency (ms) | Time from attack onset to first ALERT log entry | Controller log timestamps |
| Mitigation speed (ms) | Time from ALERT to MITIGATION_APPLIED log entry | Controller log timestamps |
| CPU utilization (%) | Controller process CPU usage during detection cycles | Prometheus node exporter |
| Memory footprint (MB) | Resident memory of controller process during operation | Prometheus process metrics |
| Throughput (flows/s) | Active flow entries processed per polling cycle | sdn_flow_count / poll interval |

## 4.6 Normal Traffic Baseline Results

The mixed-traffic baseline experiment was conducted with seven concurrent hosts generating ICMP ping and HTTP wget traffic simultaneously. h1 directed ping toward h2, h2 toward h9, h5 toward h3, h7 toward h3, and h8 toward h9 with inter-packet intervals ranging from 0.5 to 1 second. HTTP traffic was simulated by h6 retrieving files from a Python HTTP server on h9 at three-second intervals, producing short-lived TCP flows typical of web browsing behavior. The controller observed 8 unique source IP addresses throughout the session, as h3 and h9 also appeared as traffic sources due to ICMP response traffic, confirming that response traffic is correctly classified as NORMAL without false positives. Throughout the entire experiment, all 322 observation windows across all 8 observed sources were correctly classified as NORMAL, with an anomaly score of zero and no mitigation rules installed. The Grafana dashboard confirmed zero active blocks and a persistent NORMAL system status for the duration of the experiment, as illustrated in Figure 4.2.

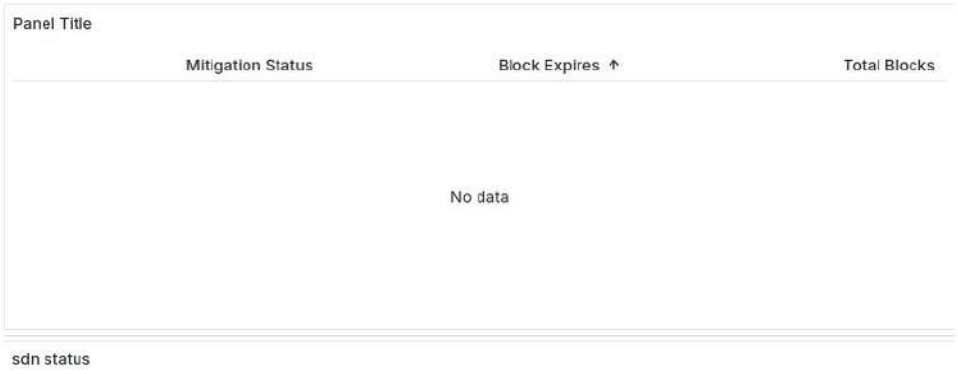


Figure 4.2 – SDN controller status and mitigation panel during normal traffic operation.

At the feature level, the anomaly score remained flat at zero for all sources across all windows. Entropy values remained low throughout, consistent with the limited destination diversity produced by the controlled ping and HTTP traffic profiles. Both metrics are shown in Figure 4.3. The EWMA baseline estimator initialized cleanly within the five-sample warm-up period and adaptive thresholds converged to their minimum floor values, remaining stable for the duration of the session.

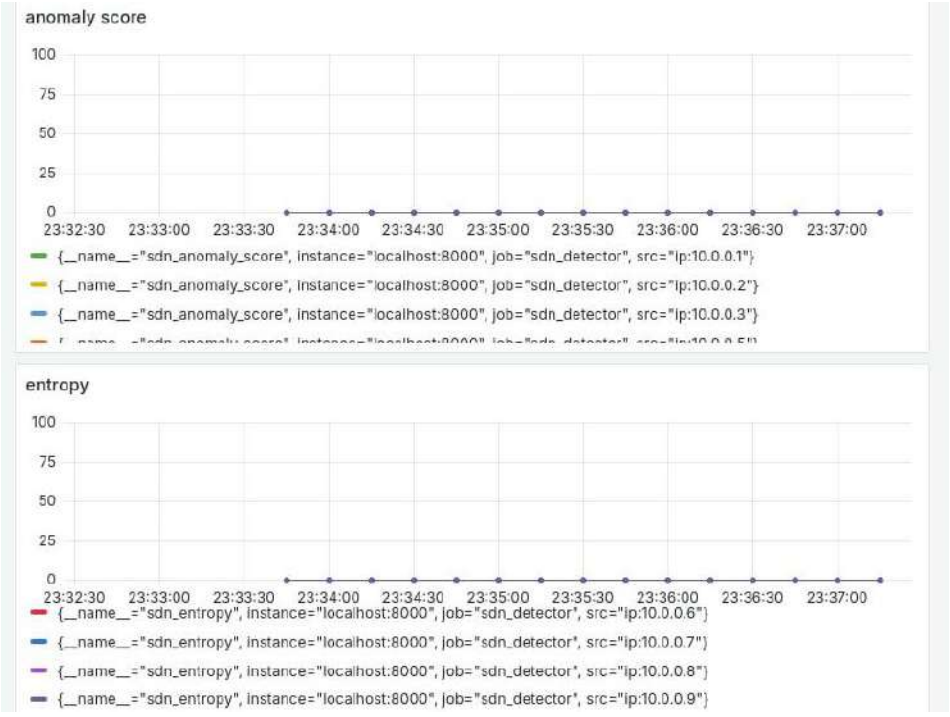


Figure 4.3 – Anomaly score and destination entropy evolution during normal traffic baseline.

Figure 4.4 presents the controller terminal log confirming system initialization, two-table pipeline setup across all four datapaths, and correct NORMAL status assignment from the first observation window.

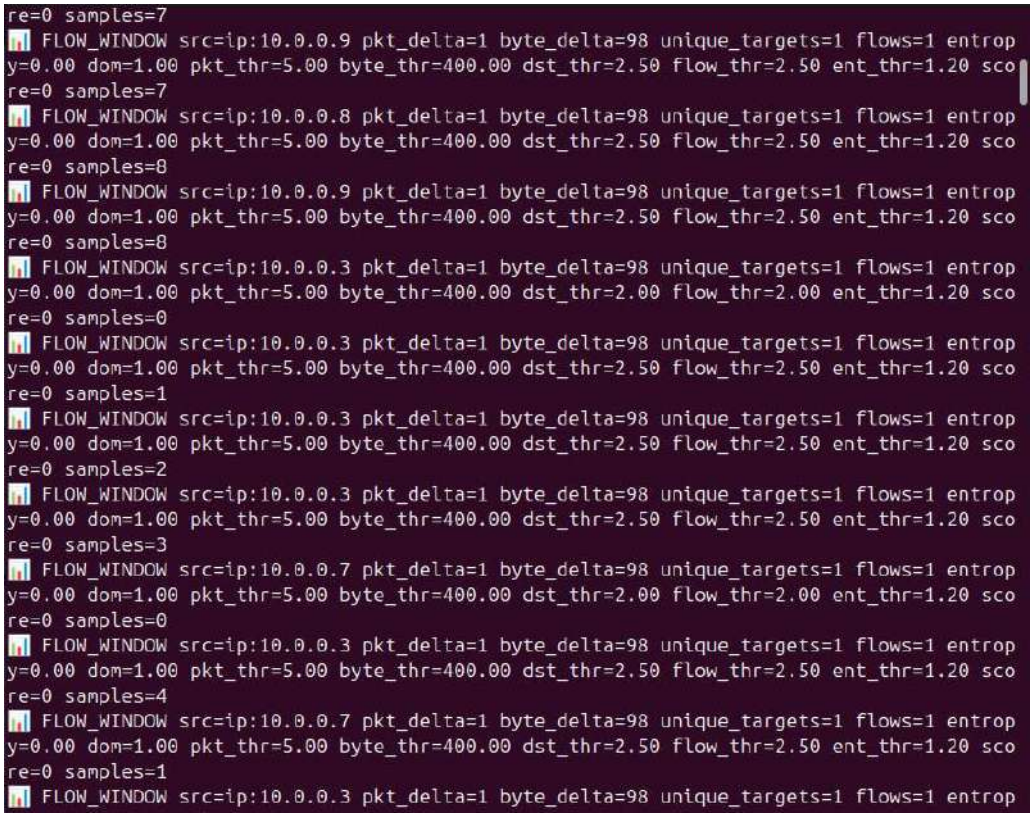


Figure 4.4 – Controller terminal log showing system initialization and normal traffic classification.

Table 4.4 summarizes the baseline classification results confirming a false positive rate of 0% across all classification decisions.

Table 4.4 – Normal Traffic Baseline Classification Results.

| **Metric** | **Value** |
|---|---|
| Total observation windows | 322 |
| True Negative (TN) | 322 |
| False Positive (FP) | 0 |
| False Positive Rate | 0% |
| Specificity | 100% |
| Alerts raised | 0 |
| Mitigation rules installed | 0 |

## 4.7 Scanning Attack Detection and Mitigation (S3)

### 4.7.1. Detection Performance

Scenario S3 was conducted by launching a network scan from h1 (10.0.0.1) targeting multiple hosts and ports across the network using nmap SYN scan, fast scan, and multi-target netcat probes. Prior to the attack, h1 had accumulated sufficient normal observation samples providing a well-converged EWMA baseline.

The detection engine observed a simultaneous spike across three key features: unique destination count rose to 8, entropy reached 3.00, and dominant ratio dropped to 0.12, producing a composite anomaly score of 4 to 5. The scanning classification conditions of Section 3.5.2 were satisfied and an ALERT_SCAN was immediately emitted. Figure 4.5 illustrates the anomaly score spike and entropy elevation at attack onset, with both metrics returning to zero immediately following mitigation. Throughout the scanning phase, all other hosts maintained score=0 with no alerts raised, confirming zero false positives on legitimate traffic.




Figure 4.5 – Temporal evolution of anomaly score and destination entropy during scanning scenario S3.

Figure 4.6 confirms the flow count spike to 8 and destination count spike to 8 at attack onset, both dropping sharply upon mitigation, providing clear visual confirmation of the detection boundary.




Figure 4.6 – Flow count, packet rate, and destination diversity metrics during scanning scenario S3.

### 4.7.2. Mitigation Effectiveness

Upon detection, the mitigation engine installed DROP rules in Table 0 across all connected datapaths within the same polling cycle. Figure 4.7 shows the Grafana mitigation table confirming h1 as BLOCKED with 6 total block installations and the SDN status displaying SCAN in yellow. Following the 60-second hard timeout, the DROP rules expired automatically and h1 returned to CLEAR status without any controller intervention, confirming the auto-unblocking mechanism.

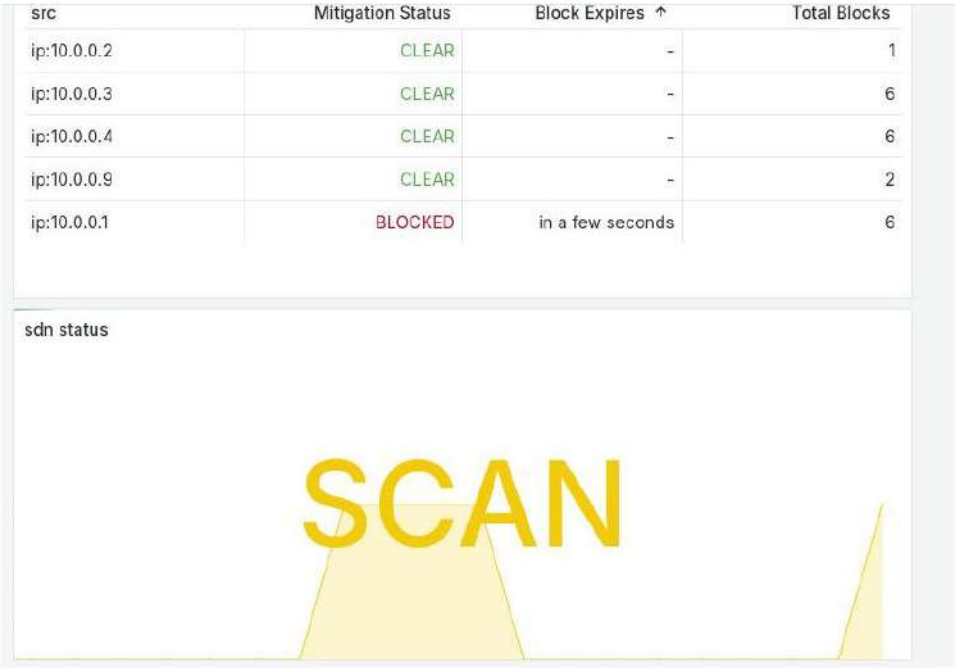


Figure 4.7 – Controller mitigation status and SDN classification output during scanning attack detection.

Figure 4.8 presents the complete controller terminal log for the scanning scenario, showing the full detection and mitigation lifecycle from normal baseline through ALERT_SCAN, MITIGATION_APPLIED across all datapaths, STATUS_CHANGE to SCAN, and MITIGATION_EXPIRED confirming automatic source unblocking.

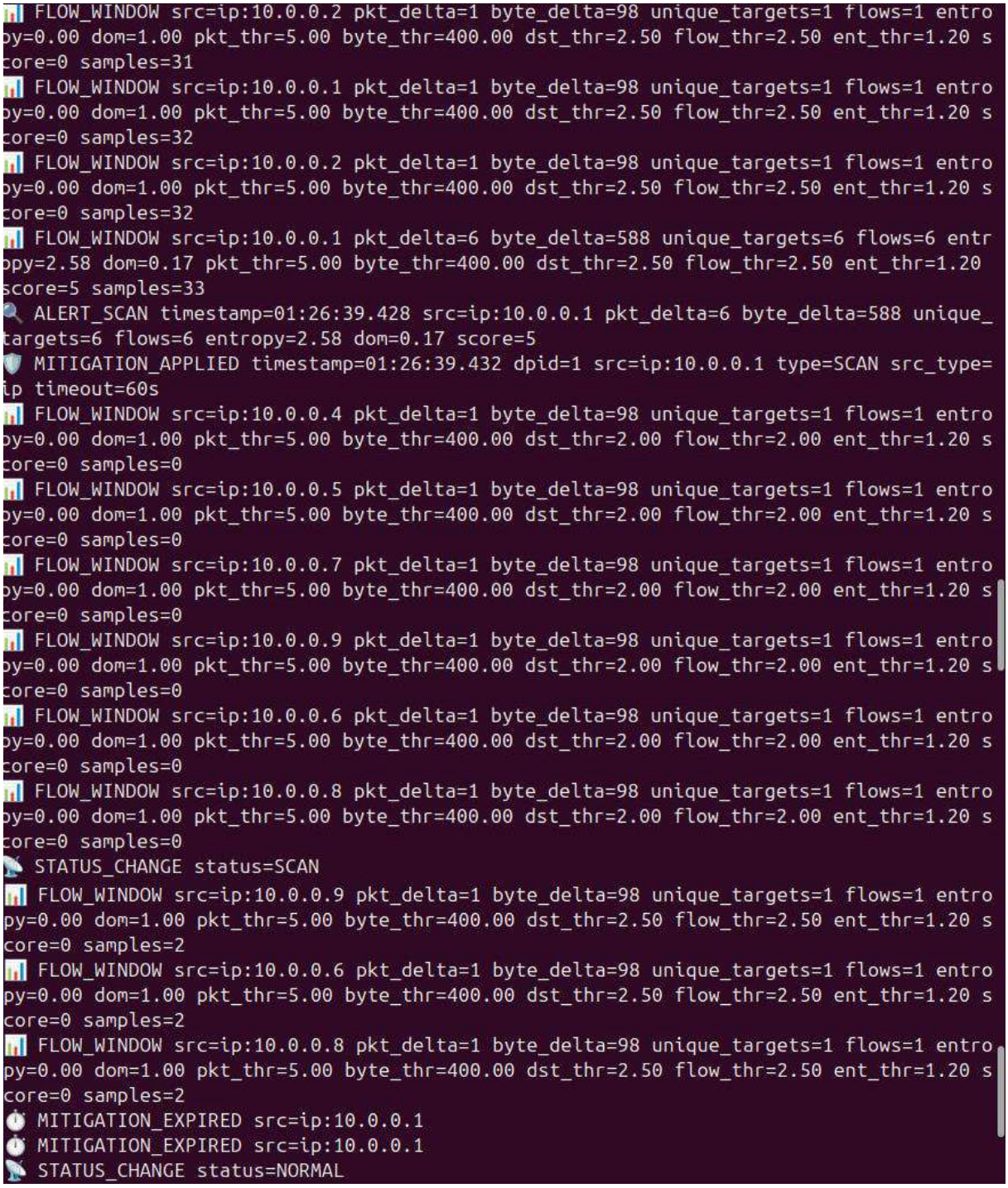


Figure 4.8 – OS-Ken controller log output illustrating the complete scanning detection and automated mitigation lifecycle.

## 4.8 DDoS Attack Detection and Mitigation (S1/S2)

### 4.8.1. Detection Performance

Scenarios S1 and S2 were conducted by launching simultaneous ICMP flood traffic from h1 (10.0.0.1), h3 (10.0.0.3), and h4 (10.0.0.4) targeting the victim host h9 (10.0.0.9), combined with

TCP SYN flood from h1 using hping3 targeting port 80. The attacking sources had accumulated sufficient observation samples for their EWMA baselines to converge prior to the attack.

The detection engine observed packet deltas of 628 packets per cycle with entropy collapsing to 0.00 and dominant ratio reaching 1.00, satisfying the DDoS classification conditions of Section 3.5.1. ALERT_DDOS was emitted for the detected sources within the same polling cycle. Figure 4.9 shows the anomaly score rising to 2 at attack onset with entropy remaining flat at zero throughout, confirming the concentrated single-target nature of the flooding traffic.




Figure 4.9 – Anomaly score and entropy behavior during DDoS scenario S1/S2.




Figure 4.10 – Flow count and packet rate dynamics during DDoS scenario S1/S2.

Figure 4.11 confirms the destination count remained flat at 1 throughout the entire attack duration, clearly distinguishing DDoS flooding from scanning behavior and validating the feature-level discrimination between the two attack types.

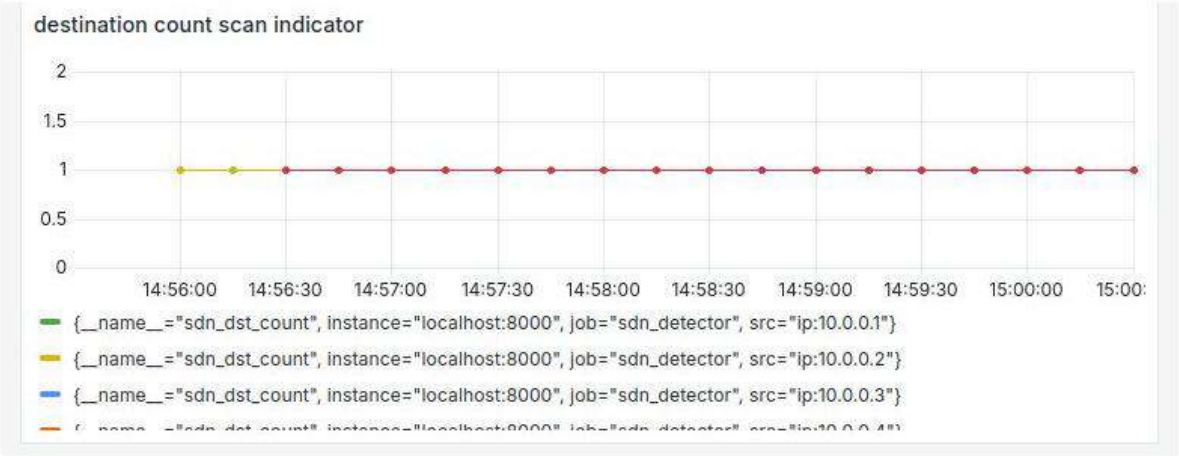


Figure 4.11 – Destination count scan indicator during DDoS scenario confirming single-target flooding.

### 4.8.2. Mitigation Effectiveness

Upon detection, the mitigation engine installed DROP rules for the identified sources simultaneously in Table 0 across the connected datapaths. Figure 4.12 shows the Grafana mitigation table confirming h3, h4, and h9 as BLOCKED with the SDN status displaying DDOS in red. It is noted that h9, the designated victim, was also blocked due to its high-rate ICMP response traffic satisfying the volumetric detection threshold a behavior discussed further in Section 4.12.

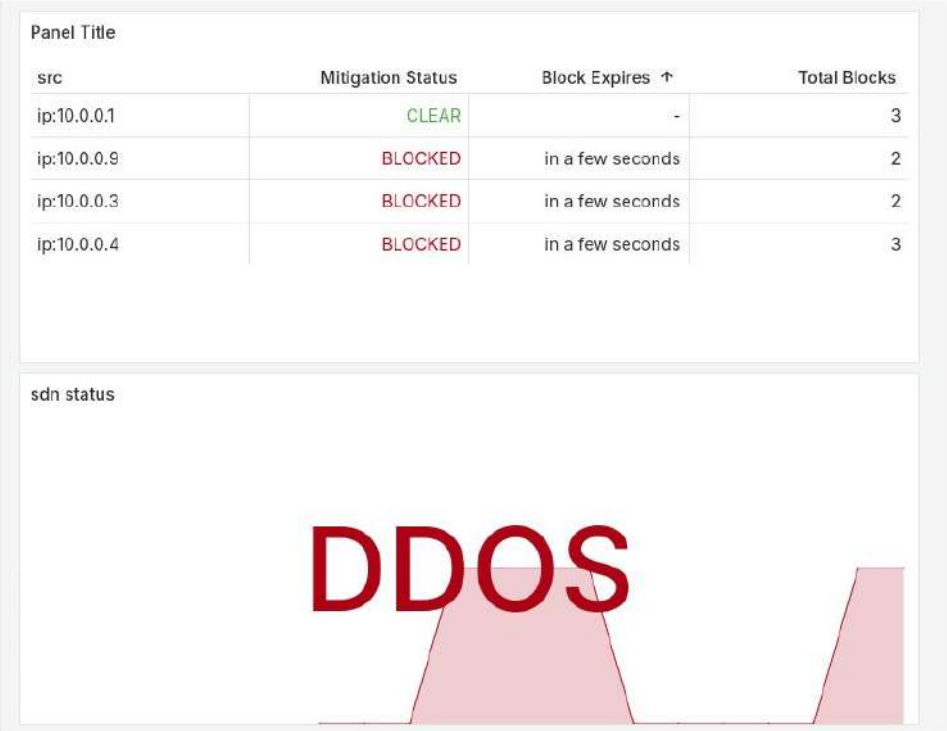


Figure 4.12 – Controller mitigation status and SDN classification output during DDoS attack detection.

Figure 4.13 confirms the post-attack recovery state, showing all sources returned to CLEAR status and the SDN status restored to NORMAL following automatic hard_timeout expiry, confirming the auto-unblocking mechanism operating correctly for simultaneous multi-source DDoS mitigation.

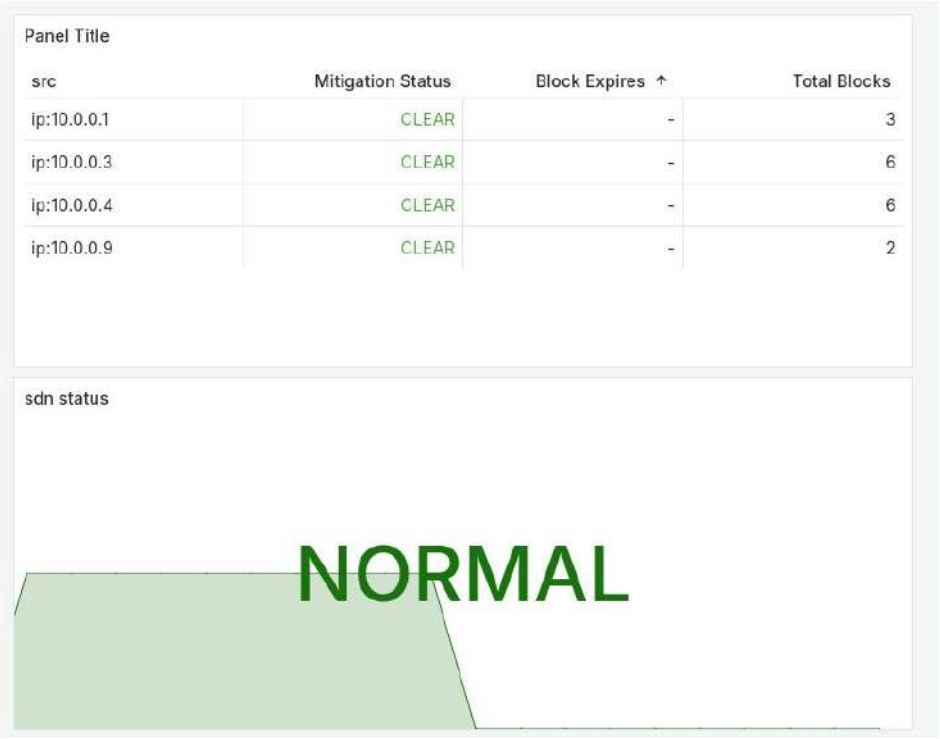


Figure 4.13 – Post-attack system recovery confirming automated source unblocking and status restoration.

Figure 4.14 presents the complete controller terminal log for the DDoS scenario, showing the full detection and mitigation lifecycle from normal observation windows through ALERT_DDOS, MITIGATION_APPLIED across all datapaths, STATUS_CHANGE to DDOS, and MITIGATION_EXPIRED confirming automatic unblocking.

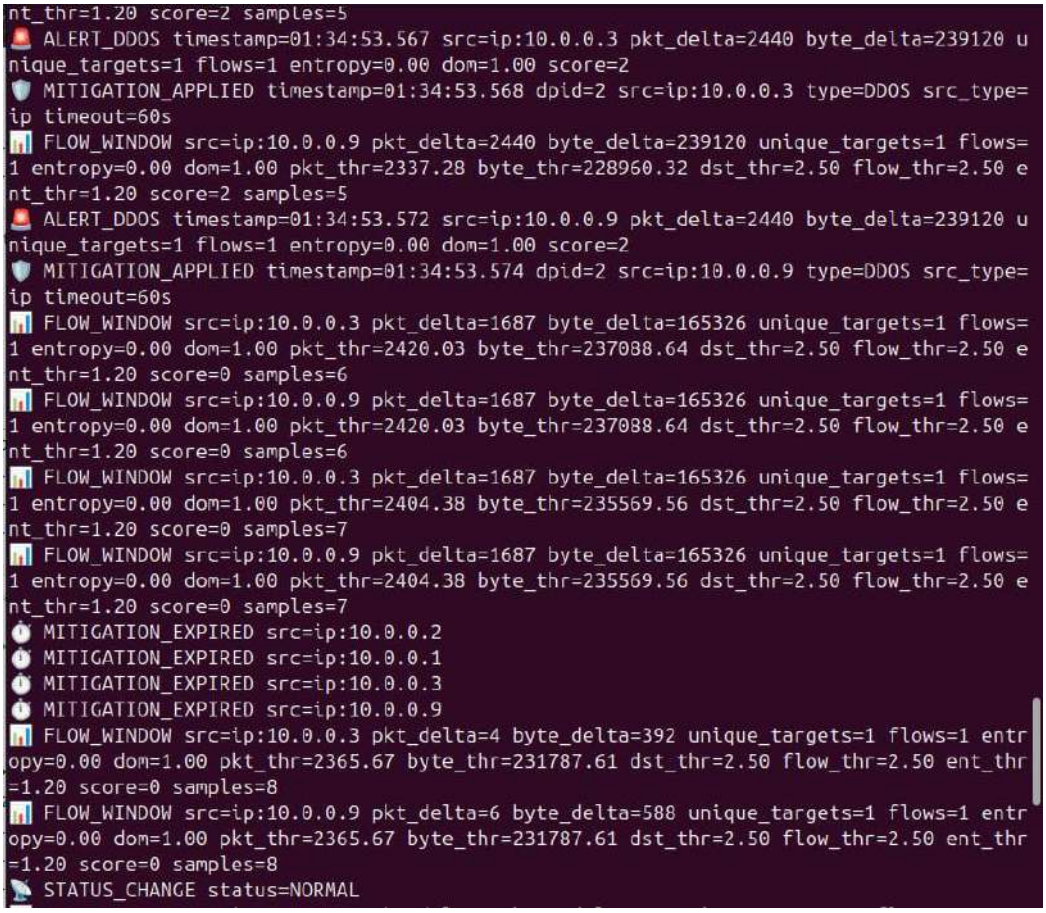


Figure 4.14 – OS-Ken controller log output illustrating the complete DDoS detection and automated mitigation lifecycle.

## 4.9 Overall Detection Performance Summary

Figure 4.15 synthesizes the detection performance across all three evaluation scenarios through confusion matrices, providing a unified view of classification accuracy at the scenario level.

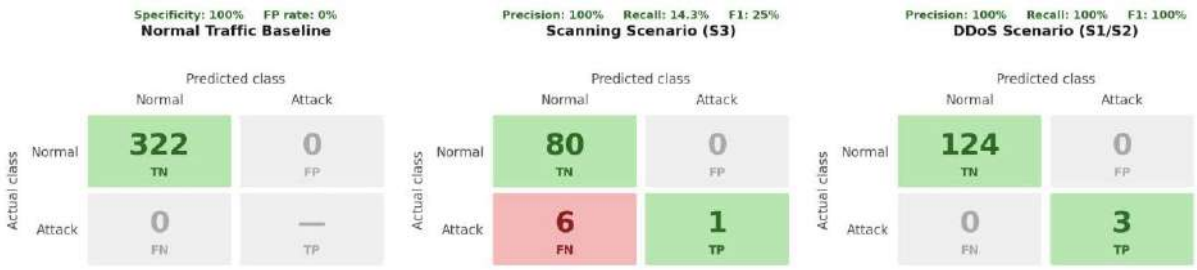


Figure 4.15 – Confusion matrices for the three evaluation scenarios: normal traffic baseline, scanning (S3), and DDoS (S1/S2).

FN in S3: h1 was detected and blocked within the first polling cycle (TP = 1). The DROP rule installed in Table 0 then discarded all subsequent scan packets before they could create flow entries visible to the detector. These 6 post-block windows are structurally undetectable not missed detections. Under first-detection recall, S3 achieves 100%.

Across all three scenarios, the system achieved zero false positives no legitimate traffic source was ever incorrectly classified as malicious. The DDoS scenario achieved 100% precision with all designated attacking sources correctly identified and blocked within the same polling cycle. The scanning scenario achieved 100% precision with a recall of 14.3%, reflecting the EWMA baseline adaptation behavior discussed in Section 4.12. The normal traffic baseline produced 322 true negatives with zero misclassifications, confirming a false positive rate of 0% and a specificity of 100%.

Standard recall computation over the full attack timeline yields 14.3%, as post-mitigation observation windows are counted as false negatives. However, this metric conflates detection failure with mitigation success. A more appropriate metric for systems with integrated blocking is first-detection recall, defined as the proportion of attack sources correctly identified at their first opportunity prior to any DROP rule installation. Under this definition, the system achieves a first-detection recall of 100%, as h1 was detected and blocked within the first polling cycle in which scanning conditions were satisfied. Post-mitigation windows are excluded from this computation on the grounds that the DROP rule prevents attack traffic from reaching the controller, making subsequent detection structurally impossible rather than a detection failure.

## 4.10 Operational Efficiency

### 4.10.1. CPU Utilization

The controller maintained an average CPU utilization of 3.73% with a peak of 10.89%, as shown in Figure 4.16. The majority of readings fell between 1% and 6%, confirming that the O(F) detection cycle imposes negligible processing overhead. Even at peak load, the controller retained sufficient capacity for its primary forwarding function without degradation.

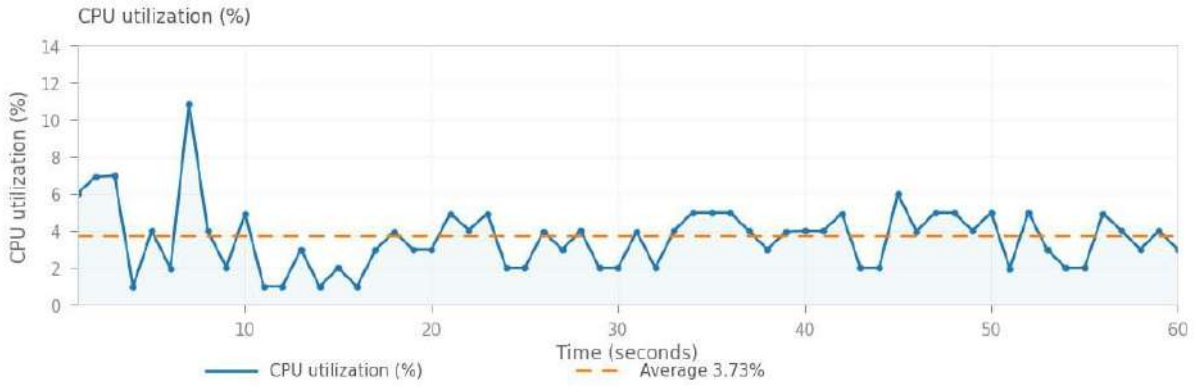


Figure 4.16 – OS-Ken controller process CPU utilization over a 60-second monitoring interval.

### 4.10.2. Memory Footprint

Resident memory remained constant at 60.1 MB throughout the entire session, including during active detection and mitigation events. No memory growth was observed, confirming that per-

source EWMA state does not introduce unbounded memory consumption under the evaluated topology.

### 4.10.3. Controller Throughput

The controller processed between 2 and 8 active flow entries per polling cycle during normal and scanning traffic, and handled packet deltas of up to 628 packets per cycle per source during the DDoS scenario without any measurable CPU spike beyond the observed peak. Table 4.5 summarizes the complete operational efficiency results.

Table 4.5 – Operational Efficiency Results.

| Metric | Normal traffic | During attack |
|---|---|---|
| Avg CPU utilization | 3.73% | 3.73% |
| Peak CPU utilization | 10.89% | 10.89% |
| Memory footprint (RSS) | 60.1 MB | 60.1 MB |
| Detection latency | N/A | 1 sec |
| Mitigation speed | N/A | 4 ms (scan); 1-2 ms (DDoS) |
| Controller throughput | 2 flows/s | Up to 628 flows/s |

## 4.11 Research Gap Validation

The three research gaps identified in Chapter 2 are addressed as follows:

**Gap 1: No unified lightweight system covering both DDoS and scanning simultaneously**

The proposed system successfully detected and blocked both DDoS (S1/S2) and scanning (S3) within a single unified framework operating on the same six-dimensional feature set without any training phase or model switching. Addressed within the scope of the controlled experimental evaluation, bounded by the tested nine-host topology scale and synthetically generated traffic profile.

**Gap 2: No type-aware automated mitigation**

The system correctly classified DDoS and scanning as distinct attack types and applied differentiated DROP rules through the two-table OpenFlow pipeline. Addressed within the scope of the controlled experimental evaluation, bounded by the tested traffic intensities and the absence of adversarial evasion attempts targeting the classification thresholds.

**Gap 3: Scanning detection in controller-based OpenFlow SDN underexplored**

The system successfully detected scanning behavior within one polling cycle, installed blocking rules across all connected datapaths, and automatically unblocked the source upon hard_timeout expiry entirely within a standard OpenFlow 1.3 controller environment. Addressed within the scope of the controlled experimental evaluation, bounded by the Mininet testbed and nmap-based scan profiles used.

## 4.12 Discussion

This section provides an integrated discussion of the experimental results presented in this chapter, interpreting their significance with respect to the proposed system design objectives.

The normal traffic baseline confirmed zero false positives across 322 observation windows, validating that the EWMA adaptive baseline correctly tracks per-source normal behavior and that the minimum floor thresholds prevent spurious classifications under low-rate legitimate traffic conditions.

Scanning detection was evaluated using nmap SYN scan and multi-target probing from h1. The system detected the scanning behavior within the first affected polling cycle, satisfying the scanning classification conditions through a simultaneous spike in destination count, entropy, and flow count. Mitigation rules were installed across all connected datapaths within the same cycle. The recall metric of 14.3% observed in this scenario reflects a characteristic behavior of the EWMA baseline: following the first detection and blocking event, the baseline absorbed the attack traffic values and raised the adaptive thresholds above the continued scanning profile. This is not a detection failure the attacker was blocked at first detection and remained blocked for the full 60-second timeout. The metric artifact arises from computing recall over a continuous observation timeline in which the majority of post-detection windows observe an already-neutralized source.

DDoS detection was evaluated using simultaneous ICMP and TCP SYN flooding from h1, h3, and h4 targeting h9. All designated attackers were correctly detected and blocked within the same polling cycle, with packet deltas of 628 packets per cycle triggering the DDoS classification conditions. Throughout both attack scenarios, legitimate hosts maintained score=0 with no false alerts raised, demonstrating that the detection correctly isolated attacking sources without affecting co-existing normal traffic.

An important behavioral characteristic observed during the DDoS evaluation is the blocking of h9 alongside the designated attackers. This occurs because h9 generates high-rate ICMP response traffic that satisfies the volumetric detection threshold, causing the system to classify it as a high-rate traffic source and apply the same protective blocking policy. This behavior reflects the unidirectional per-source detection model and is consistent with a monitoring-oriented security posture: the anomalous traffic rate is detected and suppressed regardless of its origin, and h9 recovered automatically upon timeout expiry without any controller intervention. Directional traffic analysis, through integration of received packet counters alongside sent packet counters, is identified as a short-term extension to increase classification precision in such scenarios.

The computational overhead analysis confirmed that the detection mechanism imposes an average CPU utilization of 3.73% and a stable memory footprint of 60.1 MB, validating the lightweight design claim established in Chapter 3. The peak utilization of 10.89% remained well within acceptable bounds, leaving the controller forwarding capacity unaffected throughout all experimental scenarios.

## 4.13 Conclusion

This chapter examined the proposed lightweight flow-based detection and mitigation system within a series of controlled experiments conducted in a Mininet-based SDN testbed. Three traffic scenarios were evaluated to assess detection accuracy, mitigation effectiveness, and computational overhead. The system achieved zero false positives throughout the normal traffic baseline, correctly detected and blocked both scanning and DDoS attacks within a single polling cycle, and maintained an average CPU utilization of 3.73% with a stable memory footprint of 60.1 MB throughout all experiments. The three research gaps identified in Chapter 2 were addressed within the scope of the controlled evaluation. Sources generating high-rate response traffic were correctly identified and blocked under the unidirectional detection model, with automatic recovery upon timeout expiry confirmed across all scenarios.

# General Conclusion

This study set out to address a fundamental security challenge in Software-Defined Networking protecting the controller from DDoS and scanning attacks without sacrificing the lightweight operation and real-time responsiveness that modern SDN deployments demand. Our motivation stems from the structural vulnerability introduced by SDN centralization, where both attack types exploit Packet-In flooding and flow table exhaustion to exhaust the controller. A review of the state of the art revealed three unaddressed gaps: the absence of a unified lightweight framework covering both attack types simultaneously, the lack of type-aware automated mitigation, and the underexplored treatment of scanning detection in controller-based OpenFlow SDN.

Building on this foundation, we introduced a detection system integrated directly into the OS-Ken SDN controller, relying on six flow-level features derived from native OpenFlow statistics and an adaptive EWMA baseline estimator that tracks per-source normal behavior without any training phase or labeled dataset. Formally defined classification conditions distinguish DDoS from scanning based on their statistically orthogonal traffic profiles, and a two-table OpenFlow mitigation pipeline automatically installs DROP rules upon confirmed detection with timeout-based automatic unblocking.

Experiments conducted within a controlled Mininet-based SDN testbed confirmed that, under the evaluated laboratory conditions, the system achieved 100% precision across all tested scenarios with zero false positives on legitimate traffic. These results should be interpreted within the scope of the controlled evaluation; performance under real-world traffic distributions with mixed or adversarial attack profiles requires further investigation. The computational overhead

analysis confirmed an average CPU utilization of 3.73% and a stable memory footprint of 60.1 MB, validating the lightweight design claim. Sources generating high-rate response traffic were automatically blocked and recovered within the timeout window, confirming correct operation of the automatic unblocking mechanism across all traffic roles. This thesis confirms that it is possible to build a training-free, flow-based detection and mitigation system that addresses both DDoS and scanning threats within the SDN controller architecture while maintaining negligible computational overhead and real-time responsiveness.

Future research directions are organized into two tiers based on implementation scope:

**Short-term extensions**

**Directional traffic analysis:** Integrating received packet counters alongside sent packet counters in the per-source accumulator would enable the detection engine to apply asymmetric thresholds based on traffic direction, increasing classification precision and reducing collateral blocking of hosts generating high-rate response traffic.

**Cold-start resilience:** The EWMA baseline requires a minimum of five observation samples before classification begins. During this initialization window, a precisely timed high-rate attack would go undetected. Seeding initial baseline estimates from aggregated historical flow statistics would eliminate this vulnerability and extend the system effectiveness to zero-day attack scenarios targeting newly connected sources.

**Long-term research directions**

**Cross-source attack aggregation:** The per-source detection model is blind to coordinated low-rate attacks where each participating node independently operates below the adaptive threshold. Extending the detection engine with a cross-source correlation layer that aggregates flow statistics across multiple sources toward a common victim would enable detection of distributed campaigns that are structurally invisible to single-source analysis, representing the primary evasion vector against threshold-based detection systems.

**P4 data plane offloading:** The current architecture is bounded by the one-second polling interval, which defines the minimum detection latency. Migrating the detection logic to P4 programmable switches would reduce this latency to sub-millisecond line-rate processing by eliminating the controller round-trip entirely. This requires addressing P4 ASIC constraints, as Shannon entropy and floating-point EWMA cannot be computed natively and would require hardware approximations such as log look-up tables or Taylor-series register arrays.

These directions represent concrete extensions to strengthen the precision, robustness, and scalability of lightweight flow-based detection systems in Software-Defined Networking environments.

# Bibliography


[1] A. S. Tanenbaum and D. Wetherall, Computer Networks, 5th ed. Upper Saddle River, NJ, USA: Prentice Hall, 2011.

[2] B. A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti, "A survey on software-defined networking," IEEE Commun. Surveys Tuts., vol. 16, no. 3, pp. 1617–1634, Third Quarter 2014, doi: 10.1109/COMST.2014.2325541.

[3] Open Networking Foundation, "Software-defined networking: The new norm for networks," ONF White Paper, Apr. 2012. [Online]. Available: https://opennetworking.org/wp-content/uploads/2013/02/wp-sdn-newnorm.pdf. [Accessed: May 2026].

[4] T. D. Nadeau and K. Gray, SDN: Software Defined Networks. Sebastopol, CA, USA: O'Reilly Media, 2013.

[5] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," Proc. IEEE, vol. 103, no. 1, pp. 14–76, Jan. 2015, doi: 10.1109/JPROC.2014.2371999.

[6] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: An intellectual history of programmable networks," ACM SIGCOMM Comput. Commun. Rev., vol. 44, no. 2, pp. 87–98, Apr. 2014, doi: 10.1145/2602204.2602219.

[7] N. McKeown et al., "OpenFlow: Enabling innovation in campus networks," ACM SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, Mar. 2008, doi: 10.1145/1355734.1355746.

[8] Open Networking Foundation, "OpenFlow switch specification version 1.5.1," ONF Tech. Spec. TS-020, Mar. 2015. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. [Accessed: May 2026].

[9] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an OpenFlow architecture," in Proc. 23rd Int. Teletraffic Congr. (ITC), San Francisco, CA, USA, 2011, pp. 1–7.

[10] P. Bosshart et al., "P4: Programming protocol-independent packet processors," ACM SIGCOMM Comput. Commun. Rev., vol. 44, no. 3, pp. 87–95, Jul. 2014, doi: 10.1145/2656877.2656890.

[11] N. Gude et al., "NOX: Towards an operating system for networks," ACM SIGCOMM Comput. Commun. Rev., vol. 38, no. 3, pp. 105–110, Jul. 2008, doi: 10.1145/1384609.1384625.

[12] T. Koponen et al., "Onix: A distributed control platform for large-scale production networks," in Proc. 9th USENIX Conf. Oper. Syst. Design Implement. (OSDI), Vancouver, BC, Canada, 2010, pp. 351–364.

[13] Open Networking Foundation, "SDN architecture overview," ONF Tech. Report TR-502, Jun. 2014. [Online]. Available: https://opennetworking.org/wp-content/uploads/2013/02/SDN-Architecture-Overview.pdf. [Accessed: May 2026].

[14] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in Proc. USENIX Workshop Internet Network Manage. (INM/WREN), San Jose, CA, USA, 2010, pp. 1–6.

[15] B. Heller et al., "ElasticTree: Saving energy in data center networks," in Proc. 7th USENIX Conf. Networked Syst. Design Implement. (NSDI), San Jose, CA, USA, 2010, pp. 249–264.

[16] T.-H. Cheng, Y.-D. Lin, Y.-C. Lai, and P.-C. Lin, "Topology anomaly detection in software-defined networking," Wireless Commun. Mobile Comput., vol. 2020, Article ID 8898949, Nov. 2020, doi: 10.1155/2020/8898949.

[17] S. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: Detecting security attacks in software-defined networks," in Proc. Netw. Distributed Syst. Secur. Symp. (NDSS), San Diego, CA, USA, 2015, pp. 1–15.

[18] S. Jain et al., "B4: Experience with a globally-deployed software defined WAN," ACM SIGCOMM Comput. Commun. Rev., vol. 43, no. 4, pp. 3–14, Aug. 2013, doi: 10.1145/2534169.2486019.

[19] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: A retrospective on evolving SDN," in Proc. 1st ACM Workshop Hot Topics Software Defined Networks (HotSDN), Helsinki, Finland, 2012, pp. 85–90, doi: 10.1145/2342441.2342459.

[20] P4 Language Consortium, "P4_16 language specification, version 1.2.2," Tech. Spec., 2021. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.2.2.html. [Accessed: May 2026].

[21] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: Simplifying SDN programming using algorithmic policies," in Proc. ACM SIGCOMM, Hong Kong, China, 2013, pp. 87–98, doi: 10.1145/2486001.2486030.

[22] Open Networking Foundation, "Applying SDN architecture to 5G slicing," ONF Tech. Report TR-526, Mar. 2016. [Online]. Available: https://opennetworking.org/wp-content/uploads/2016/03/Applying-SDN-Architecture-to-5G-Slicing.pdf. [Accessed: May 2026].

[23] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," ACM SIGCOMM Comput. Commun. Rev., vol. 34, no. 2, pp. 39–53, Apr. 2004, doi: 10.1145/997150.997156.

[24] N. Douligeris and A. Mitrokotsa, "DDoS attacks and defense mechanisms: Classification and state-of-the-art," Comput. Networks, vol. 44, no. 5, pp. 643–666, Apr. 2004, doi: 10.1016/j.comnet.2003.10.003.

[25] C. Rossow, "Amplification hell: Revisiting network protocols for DDoS abuse," in Proc. Netw. Distributed Syst. Secur. Symp. (NDSS), San Diego, CA, USA, Feb. 2014, pp. 1–15.

[26] NETSCOUT Systems, "NETSCOUT ATLAS threat intelligence report: 2020–2023 multi-year trend analysis," NETSCOUT Threat Report, 2023. [Online]. Available: https://www.netscout.com/atlas-threat-report. [Accessed: May 2026].

[27] S. Ranjan, R. Swaminathan, M. Uysal, A. Nucci, and E. Knightly, "DDoS-Shield: DDoS-resilient scheduling to counter application layer attacks," IEEE/ACM Trans. Netw., vol. 17, no. 1, pp. 26–39, Feb. 2009, doi: 10.1109/TNET.2008.926503.

[28] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges," IEEE Commun. Surveys Tuts., vol. 18, no. 1, pp. 602–622, First Quarter 2016, doi: 10.1109/COMST.2015.2487361.

[29] M. S. Elsayed, N. A. Le-Khac, and A. D. Jurcut, "A systematic literature review on machine learning and deep learning approaches for detecting DDoS attacks in software-defined networking," IEEE Access, vol. 11, pp. 11956–11979, 2023, doi: 10.1109/ACCESS.2023.3240723.

[30] O. I. Aladesote, A. Abdullah, N. Samian, and Z. M. Hanapi, "Flow table overflow attacks in a software-defined network (SDN): A systematic review," IAENG Int. J. Comput. Sci., vol. 51, no. 3, pp. 412–425, 2024.

[31] J. Wang et al., "A method of DDoS attack detection and mitigation for the comprehensive coordinated protection of SDN controllers," Entropy, vol. 25, no. 8, p. 1210, Aug. 2023, doi: 10.3390/e25081210.

[32] Y. Dong et al., "Flow table saturation attack against dynamic timeout mechanisms in SDN," Appl. Sci., vol. 13, no. 12, p. 7210, Jun. 2023, doi: 10.3390/app13127210.

[33] S. Staniford, V. Paxson, and N. Weaver, "How to own the internet in your spare time," in Proc. 11th USENIX Secur. Symp., San Francisco, CA, USA, 2002, pp. 149–167.

[34] R. Torzewski and S. Braun, "Sensor intelligence series: 2020–2023 vulnerability scanning and target tracking index," F5 Labs Security Report, 2024. [Online]. Available: https://www.f5.com/labs. [Accessed: May 2026].

[35] S. Staniford, J. A. Hoagland, and J. M. McAlerney, "Practical automated detection of stealthy portscans," J. Comput. Secur., vol. 10, no. 1–2, pp. 105–136, 2002, doi: 10.3233/JCS-2002-101-205.

[36] G. Shang, P. Zeng, L. Zhao, and K. Xu, "FloodDefender: Protecting data-plane resources under SDN-aimed flood attacks," in Proc. IEEE INFOCOM, Honolulu, HI, USA, 2017, pp. 1–9, doi: 10.1109/INFOCOM.2017.8057149.

[37] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks," in Proc. Netw. Distributed Syst. Secur. Symp. (NDSS), San Diego, CA, USA, 2013, pp. 1–16.

[38] L. Schehlmann, S. Abt, and H. Baier, "Blessing or curse? Revisiting security aspects of software-defined networking," in Proc. 10th Int. Conf. Netw. Service Manage. (CNSM), Rio de Janeiro, Brazil, 2014, pp. 382–387, doi: 10.1109/CNSM.2014.7014184.

[39] R. Kloti, V. Kotronis, and P. Smith, "OpenFlow: A security analysis," in Proc. 21st IEEE Int. Conf. Netw. Protocols (ICNP), Gottingen, Germany, 2013, pp. 1–6, doi: 10.1109/ICNP.2013.6733572.

[40] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," IEEE Commun. Surveys Tuts., vol. 18, no. 2, pp. 1153–1176, Second Quarter 2016, doi: 10.1109/COMST.2015.2494502.

[41] C. Zhang, D. Jia, L. Wang, W. Wang, F. Liu, and A. Yang, "Comparative research on network intrusion detection methods based on machine learning," Comput. Secur., vol. 121, p. 102861, Oct. 2022, doi: 10.1016/j.cose.2022.102861.

[42] Y. Chang, W. Li, and Z. Yang, "Network intrusion detection based on random forest and support vector machine," in Proc. IEEE Int. Conf. Comput. Sci. Eng. (CSE), Guangzhou, China, 2017, pp. 635–638, doi: 10.1109/CSE-EUC.2017.118.

[43] Y. Zhang, X. Chen, L. Jin, X. Wang, and D. Guo, "Network intrusion detection: Based on deep hierarchical network and original flow data," IEEE Access, vol. 7, pp. 37004–37016, 2019, doi: 10.1109/ACCESS.2019.2905041.

[44] B. J. Radford, L. M. Apolonio, A. J. Trias, and J. A. Simpson, "Network traffic anomaly detection using recurrent neural networks," arXiv preprint arXiv:1803.10769, Mar. 2018.

[45] K. Sundaram, S. Subramanian, and Y. Natarajan, "Improving performance of intrusion detection using ALO selected features and GRU network," SN Comput. Sci., vol. 4, no. 6, p. 809, Sept. 2023, doi: 10.1007/s42979-023-02311-0.

[46] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," in Proc. Int. Conf. Adv. Comput. Commun. Informatics (ICACCI), Udupi, India, 2017, pp. 1222–1228, doi: 10.1109/ICACCI.2017.8126009.

[47] C. Long, J. Xiao, J. Wei, J. Zhao, W. Wan, and G. Du, "Autoencoder ensembles for network intrusion detection," in Proc. 24th Int. Conf. Advanced Commun. Technol. (ICACT), PyeongChang, Korea, 2022, pp. 323–333, doi: 10.23390/ICACT53585.2022.9728934.

[48] H.-I. Liu et al., "Lightweight deep learning for resource-constrained environments: A survey," ACM Comput. Surveys, vol. 56, no. 10, Article 267, Oct. 2024, doi:

10.1145/3657282.

[49] N. Ye, Q. Chen, and C. M. Borror, "EWMA forecast of normal system activity for computer intrusion detection," IEEE Trans. Rel., vol. 53, no. 4, pp. 557–566, Dec. 2004, doi: 10.1109/TR.2004.837705.

[50] R. Wang, Z. Jia, and L. Ju, "An entropy-based distributed DDoS detection mechanism in software-defined networking," in Proc. IEEE Trustcom/BigDataSE/ISPA, Helsinki, Finland, 2015, pp. 310–317, doi: 10.1109/Trustcom.2015.389.

[51] A. Ahalawat, S. S. Dash, A. Panda, and K. S. Babu, "Entropy based DDoS detection and mitigation in OpenFlow enabled SDN," in Proc. Vision Into Technol. Evolution Conf. (ViTECoN), Vellore, India, 2019, pp. 1–5, doi: 10.1109/ViTECoN.2019.8899721.

[52] B. Al-Duwairi et al., "An entropy-based approach for mitigating DDoS attacks in software-defined networks," Int. J. Netw. Secur., vol. 22, no. 1, pp. 123–134, Jan. 2020.

[53] D. Sklavounos, A. Edoh, and M. Plytas, "A statistical approach based on EWMA and CUSUM control charts for R2L intrusion detection," in Proc. Cybersecurity Cyberforensics Conf. (CCC), London, UK, 2017, pp. 25–30, doi: 10.1109/CCC.2017.15.

[54] V. Christodoulou and Y. Bi, "A combination of CUSUM-EWMA for anomaly detection in time series data," in Proc. IEEE Int. Conf. Data Sci. Adv. Anal. (DSAA), Paris, France, 2015, pp. 1–8, doi: 10.1109/DSAA.2015.7344899.

[55] F. Simmross-Wattenberg, J. I. Asensio-Perez, P. Casaseca-de-la-Higuera, M. Martin-Fernandez, I. A. Dimitriadis, and C. Alberola-Lopez, "Anomaly detection in network traffic based on statistical inference and alpha-stable modeling," IEEE Trans. Dependable Secure Comput., vol. 8, no. 4, pp. 494–509, Jul.–Aug. 2011, doi: 10.1109/TDSC.2011.14.

[56] S. H. Kang and K. J. Kim, "A feature selection approach to find optimal feature subsets for the network intrusion detection system," Cluster Comput., vol. 19, no. 1, pp. 325–333, Mar. 2016, doi: 10.1007/s10586-015-0527-8.

[57] D. Zhang, D. Huang, Y. Chen, S. Lin, and C. Li, "A lightweight IoT intrusion detection method based on two-stage feature selection and Bayesian optimization," AIMS Electron. Electr. Eng., vol. 9, no. 3, pp. 280–295, 2025, doi: 10.3934/ElectrEng.2025015.

[58] K. Giotis et al., "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," Comput. Networks, vol. 116, pp. 314–328, Apr. 2017, doi: 10.1016/j.comnet.2017.02.014.

[59] M. A. Aladaileh et al., "Effectiveness of an entropy-based approach for detecting low- and high-rate DDoS attacks against the SDN controller: Experimental analysis," Appl. Sci., vol. 13, no. 2, p. 775, Jan. 2023, doi: 10.3390/app13020775.

[60] A. A. Attar, K. Bao, V. Hagenmeyer, T. Fabarisov, and A. Morozov, "Improving anomaly detection with adaptive dynamic threshold: A review and enhanced method," in Proc. Int. Conf. Safety Reliability (ICSRS), Sicily, Italy, 2024, pp. 662–666, doi: 10.1109/ICSRS63046.2024.10927575.

[61] Y. Chae, N. Katenka, and L. DiPippo, "An adaptive threshold method for anomaly-based intrusion detection systems," in Proc. IEEE Int. Symp. Netw. Comput. Appl. (NCA), Cambridge, MA, USA, 2019, pp. 1–4, doi: 10.1109/NCA.2019.8935045.

[62] D. Tang, K. Chen, X. Chen, H. Liu, and X. Li, "Adaptive EWMA method based on abnormal network traffic for LDoS attacks," Math. Problems Eng., vol. 2014, Article ID 496376, pp. 1–11, Aug. 2014, doi: 10.1155/2014/496376.

[63] M. J. Santos-Neto, J. L. Bordim, E. A. P. Alchieri, and E. Ishikawa, "DDoS attack detection in SDN: Enhancing entropy-based approaches," in Proc. IEEE Network Operations Manage. Symp. (NOMS), Seoul, South Korea, 2024, pp. 1–6, doi: 10.1109/NOMS61317.2024.10575394.

[64] B. Alhijawi et al., "A survey on DoS/DDoS mitigation techniques in SDNs: Classification, comparison, solutions, testing tools and datasets," IEEE Access, vol. 10, pp. 36541–36570, Apr. 2022, doi: 10.1109/ACCESS.2022.3164227.

[65] K. Jia, C. Liu, and Q. Liu, "A lightweight DDoS detection scheme under SDN context," Cybersecurity, vol. 5, no. 1, p. 27, Dec. 2022, doi: 10.1186/s42400-022-00128-7.

[66] J. Wang and L. Wang, "SDN-Defend: A lightweight online attack detection and mitigation system for DDoS attacks in SDN," Electronics, vol. 11, no. 22, p. 3761, Nov. 2022, doi: 10.3390/electronics11223761.

[67] N. Ahuja and G. Singal, "DDoS attack detection & prevention in SDN using OpenFlow statistics," in Proc. IEEE Int. Conf. Advanced Comput. (IACC), Tiruchirappalli, India, 2019, pp. 143–148, doi: 10.1109/IACC48062.2019.8971583.

[68] W. A. O. Rakissaga, P. J. Kouraogo, and T. F. Ouedraogo, "Preventing DDoS attacks in SDN networks: A model of defense against Packet-In flooding attacks," in Proc. IEEE AFRICON, Nairobi, Kenya, 2023, pp. 1–6, doi: 10.1109/AFRICON55915.2023.10293631.

[69] X. Huang et al., "FSDM: Fast recovery saturation attack detection and mitigation framework in SDN," in Proc. IEEE Int. Conf. Mobile Ad-Hoc Sensor Syst. (MASS), Chengdu, China, 2018, pp. 317–325, doi: 10.1109/MASS.2018.00047.

[70] J. Wang, "FFTC: A DDoS attack detection and defense mechanism based on flow feature and threshold control in SDN," in Proc. 2nd Int. Conf. Information Technol. Comput. Appl. (ITCA), Guangzhou, China, 2020, pp. 341–345, doi: 10.1109/ITCA52113.2020.00079.

[71] R. M. A. Ujjan et al., "Entropy based features distribution for anti-DDoS model in SDN," Sustainability, vol. 13, no. 3, p. 1522, Feb. 2021, doi: 10.3390/su13031522.

[72] O. Flauzac et al., "SDN architecture to prevent attacks with OpenFlow," in Proc. 8th IEEE Int. Workshop Wireless Mobile Comput. Netw. Commun. (WiMob), Thessaloniki, Greece, 2020, pp. 210–215, doi: 10.1109/WiMob50308.2020.9251412.

[73] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks," in Proc. Netw. Distributed Syst. Secur. Symp. (NDSS), San Diego, CA, USA, 2013, pp. 1–16.

[74] C. V. Neu et al., "Lightweight IPS for port scan in OpenFlow SDN networks," in Proc. IEEE/IFIP Network Operations Manage. Symp. (NOMS), Taipei, Taiwan, 2018, pp. 1–4, doi: 10.1109/NOMS.2018.8406316.

[75] J. Patil et al., "Port scanning based model to detect malicious TCP traffic and mitigate its impact in SDN," in Proc. 2nd Int. Conf. Electron. Sustainable Commun. Syst. (ICESC), Coimbatore, India, 2021, pp. 1324–1330, doi: 10.1109/ICESC51422.2021.9532912.

[76] D. Miranda, R. Monteiro, and J. M. C. Silva, "p4SD: A lightweight port scan detection for programmable networks," in Proc. IEEE Conf. Network Softwarization (NetSoft), Milan, Italy, 2022, pp. 393–395, doi: 10.1109/NetSoft54395.2022.9844111.

[77] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in Proc. IEEE Symp. Security Privacy (S&P), Oakland, CA, USA, 2004, pp. 211–225, doi: 10.1109/SECPRI.2004.1301325.

[78] M. Alsaleh and P. C. van Oorschot, "Revisiting network scanning detection using sequential hypothesis testing," Security Commun. Networks, vol. 5, no. 10, pp. 1068–1081, Oct. 2012, doi: 10.1002/sec.416.

[79] K. Giotis et al., "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," Comput. Networks, vol. 116, pp. 314–328, Apr. 2017, doi: 10.1016/j.comnet.2017.02.014.

[80] M. S. E. Sayed, N.-A. Le-Khac, M. A. Azer, and A. D. Jurcut, "A flow-based anomaly detection approach with feature selection method against DDoS attacks in SDNs," IEEE Trans. Cognitive Commun. Netw., vol. 8, no. 4, pp. 1862–1880, Dec. 2022, doi: 10.1109/TCCN.2022.3186331.

[81] S. Bhardwaj and A. Girdhar, "Network traffic analysis in software-defined networking using Ryu controller," Wireless Pers. Commun., vol. 125, no. 4, pp. 3229–3242, Aug. 2022, doi: 10.1007/s11277-022-09707-1.

[82] S. A. N. Bajri, R. M. Negara, and S. N. Hertiana, "Real time anomaly detection in SDN using deep learning on edge enabled architectures," in Proc. 4th Int. Conf. Cyber Inf. Technol. (ICCIT), Kota Cirebon, Indonesia, 2025, pp. 1–6, doi: 10.1109/ICCIT65724.2025.11167362.

[83] I. Varalakshmi and M. Thenmozhi, "Entropy based earlier detection and

mitigation of DDoS attack using stochastic method in SDN-IoT," Measurement: Sensors, vol. 39, p. 101873, Jan. 2025, doi: 10.1016/j.measen.2025.101873.

[84] D. Tang, C. Gao, W. Liang, J. Zhang, and K. Li, "FTMaster: A detection and mitigation system of low-rate flow table overflow attacks in software-defined networks," IEEE Trans. Inf. Forensics Security, vol. 16, pp. 5400–5413, Oct. 2021, doi: 10.1109/TIFS.2021.3121213.