**IMT School for Advanced Studies, Lucca**
Lucca, Italy

# Innovative Deep Learning Methods and a Benchmarking Dataset for Intrusion Detection Systems in Software-Defined Networks

PhD Program Cybersicurezza

Track in Software, System, and Infrastructure Security

XXXVIII Cycle

**By**

**Francesco Di Gennaro**

**2026**

**The dissertation of Francesco Di Gennaro is approved.**

PhD Program Coordinator: Prof. Mirco Tribastone, IMT School for Advanced Studies Lucca

Advisor: Prof. Luca Spalazzi, Università Politecnica delle Marche

Co-Advisor: Prof. Alessandro Cucchiarelli, Università Politecnica delle Marche

The dissertation of Francesco Di Gennaro has been reviewed by:

Reviewer 1: Prof. Martin Gilje Jaatun, SINTEF Digital, Trondheim, Norway

Reviewer 2: Prof. Martin Pinzger, Universität Klagenfurt, Klagenfurt, Austria

IMT School for Advanced Studies Lucca
2026

# Contents

# List of Figures

# List of Tables

# Acknowledgements

This work was conducted within the broader context of the author's doctoral programme and benefited from the guidance, support and contributions of several individuals and institutions.

First, sincere gratitude is expressed to the Prof. Luca Spalazzi, Prof. Alessandro Cucchiarelli and Prof. Christian Morbidoni for continuous scientific guidance, constructive feedback and careful oversight throughout the research process. The supervisor's ability to combine methodological rigor with pragmatic judgment has been essential in shaping both the direction of the work and the clarity of its presentation. Appreciation is also extended to the members of the doctoral committee and external reviewers for their time, their critical reading and their valuable suggestions, which contributed to improving the quality and coherence of the thesis.

The author is grateful to colleagues and collaborators who provided technical support and insightful discussions during the design and execution of the experimental campaigns. Their availability and competence were particularly valuable in addressing practical challenges related to testbed configuration, data acquisition and large-scale processing workflows. This thesis also incorporates, in selected parts, material based on co-authored publications and a co-authored manuscript currently under review. In accordance with the program requirements, the relevant co-authors, venues and publishers are explicitly acknowledged below.

Section 3.1 includes text based on: Francesco Di Gennaro, Alessandro Cucchiarelli, Christian Morbidoni and

Luca Spalazzi, "A Comparative Analysis of Datasets for Intrusion Detection in Software-Defined Networks", presented at *ITASEC 2025*, Bologna, Italy, 2025, published in the workshop proceedings by *CEUR-WS.org*.

Section 3.2 includes text based on: Francesco Di Gennaro, Alessandro Cucchiarelli, Christian Morbidoni and Luca Spalazzi, "Feature selection in ML-based SDN intrusion detection system", presented at the *11th International Conference on Future Internet of Things and Cloud (FiCloud)*, Vienna, Austria, 2024, published in the conference proceedings by IEEE.

Chapter 4 includes text based on: Francesco Di Gennaro, Alessandro Cucchiarelli, Christian Morbidoni and Luca Spalazzi, "KRONOS-SDN: A Large-Scale, Cross-Plane Dataset for Machine Learning Intrusion Detection in Software-Defined Networks", a manuscript currently under review at the *Journal of Network and Computer Applications, Elsevier*.

This research benefited from access to computing resources and laboratory facilities from Department of Engineering of Information of the *Università Politecnica delle Marche,* that enabled extensive experimentation. The author acknowledges the institutions and organizational structures that supported the activities underpinning this thesis and expresses gratitude for the administrative assistance that facilitated the timely completion of several activities and milestones.

During the preparation of this thesis the author used GPT in order to review grammar and paraphrasing. After using this tool/service, the author reviewed and edited the content.

Finally, the author wishes to thank family for their patience, encouragement and unwavering support throughout the PhD path. Their understanding during demanding periods and their constant presence, have provided the stability and motivation necessary to bring this work to completion.

# Vita

| | |
|---|---|
| **January 29, 1995** | Born, Lacco Ameno (NA), Italy |
| **September 25, 2019** | Bachelor in Electronic Engineering<br>Final mark: 110/110 cum laude<br>Università degli studi di Napoli "Federico II", Naples, Italy |
| **July 21, 2021** | Master Degree in Electronic Engineering<br>Final mark: 110/110 cum laude<br>Università degli studi di Napoli "Federico II", Naples, Italy |
| **2022 - 2025** | PhD Candidate<br>PhD of National Interest in Cybersecurity<br>IMT Scuola Alti Studi Lucca, Lucca, Italy<br>Università Politecnica delle Marche, Ancona, Italy |

# Publications

1. F. Di Gennaro, A. Cucchiarelli, C. Morbidoni, L. Spalazzi, “Feature selection in ML-based SDN intrusion detection system.” Proceedings of the 11th International Conference on Future Internet of Things and Cloud (FiCloud 2024), Vienna, Austria, August 19–21, pp. 152–159, IEEE, 2024. doi: 10.1109/FiCloud62933.2024.00031. 
2. F. Di Gennaro, A. Cucchiarelli, C. Morbidoni, L. Spalazzi, “A Comparative Analysis of Datasets for Intrusion Detection in Software-Defined Networks”. CEUR Workshop Proceedings, vol. 3962, CEUR-WS, 2025.
3. F. Di Gennaro, A. Cucchiarelli, C. Morbidoni, L. Spalazzi, “KRONOS-SDN: A Large-Scale, Cross-Plane Dataset for Machine Learning Intrusion Detection in Software-Defined Networks.” Under review at Journal of Network and Computer Applications, 2025.
4. F. Di Gennaro, A. Giuliani, C. Morbidoni, A. Cucchiarelli, L. Spalazzi, ”A Hierarchical Hybrid Deep Learning Framework for Unknown Attack Detection in SDN Networks”. CEUR Workshop Proceedings, vol. 4198, CEUR-WS, 2026.

# Presentations

1. "Feature selection in ML-based SDN intrusion detection system" at *11th International Conference on Future Internet of Things and Cloud (FiCloud)*, Wien, Austria, 2024.
2. "A Comparative Analysis of Datasets for Intrusion Detection in Software-Defined Networks" at *ITASEC2025*, Bologna, Italy, 2025.

# Abstract

Software-Defined Networking (SDN) improves programmability and operational agility by separating the control plane from the forwarding one and by relying on a logically centralized controller. This architectural shift also concentrates risk: the controller ecosystem becomes a high-value target and a potential bottleneck, so security monitoring must be effective and remain feasible under near-real-time constraints. A recurring limitation in the SDN Intrusion Detection System (IDS) literature is that many results are obtained on benchmarks that provide limited temporal continuity, simplified operational dynamics or insufficient multi-plane observability, which complicates reproducibility and weakens deployment-oriented conclusions.

This thesis tackles the above gap through both a dataset and methodological contribution. First, it derives concrete benchmarking requirements from a structured comparison of SDN intrusion datasets and evaluation practices, then it introduces *KRONOS-SDN*, a reproducible SDN IDS benchmark dataset designed to support controlled experiments over temporally coherent traces and cross-plane signals. KRONOS-SDN aligns flow-derived representations with controller host telemetry, enabling the investigation of detection behavior under realistic workload variation. Second, the thesis develops a deep-learning–based IDS that infers on sliding temporal windows of entity-level feature vectors, enabling streaming-compatible analysis and presents a controlled comparison of representative reconstruction-based models (CNN, LSTM, Transformer and MLP-based autoencoders) under homogeneous training, calibration and scoring assumptions. In addition to detection

effectiveness, the study quantifies operational indicators such as alert latency, throughput and computational footprint.

The results show that deep anomaly detection can achieve meaningful detection capability on realistic SDN traces, but also that alert stability is sensitive to legitimate operating-regime changes. To address this issue and to increase the IDS performances, the third contribution of the thesis consists of proposing a lightweight cross-plane strategy based on *context modulation*: controller-side telemetry is used to infer the current operating regime and to adapt the detection threshold over time, reducing workload-induced false positives while preserving responsiveness to related deviations to attacks. Overall, the thesis advances SDN IDS evaluation toward realistic, reproducible and deployment-oriented conclusions by jointly contributing an enabling benchmark dataset and an operationally grounded methodology for near-real-time anomaly detection.

# Chapter 1

# Introduction

## 1.1 Motivation

Software-Defined Networking (SDN) has become a key architectural approach for achieving greater programmability and operational flexibility in contemporary networks, primarily through decoupling forwarding from control logic and enabling policy enforcement through a logically centralized control function. The movement towards a logically centralized control function provides several benefits such as increased global visibility and enhanced automation however it also changes the nature of the security surface. Accordingly, the centrally located controller, and all of the services that support the controller, along with the control channels used to connect the various elements of the architecture represent a highly valuable target for attackers, and their compromise, overloading, or misconfiguration can have significant impact on the overall operation of the network. In this setting, intrusion detection represents a fundamental capability required to ensure availability and correctness within the network under adversarial conditions.

The application of machine learning to detect intrusions has been extensively studied as a mechanism to move beyond signature based systems and to enable detection of complex and dynamic malicious behaviour. A significant challenge encountered by researchers studying

SDN-based intrusion detection systems (IDS), is a methodological gap between the development of sophisticated detection methodologies and the empirical validation of those methods. Specifically, detection methods are commonly evaluated against a set of experimental datasets that inadequately model SDN-specific behaviour, lack temporal consistency, or do not adequately document the ground truth associated with the experimental results. As a result, seemingly successful evaluations may be attributable to artifacts within the experimental dataset, and not to the ability of the detection methodology to identify malicious activity in a deployed environment. Therefore, prior to advocating for the use of increasingly complex detection architectures, there is a need to critically evaluate the evidence base and to develop controlled and reproducible protocols that jointly evaluate the detection quality of a given methodology and the operational constraints that would affect deployment.

Another motivation for this thesis is to investigate the problem of observability across planes and domains in SDNs. Realistic SDN deployments are characterized by feedback loops between the data plane and the controller plane. For example, anomalous traffic patterns can negatively impact the resources available to the control plane, and the operating regime of the controller can influence the types of anomalies that are observable and at what level of granularity. Nevertheless, cross-plane fusion has received relatively little attention in the SDN literature due in part to the fact that very few benchmarking platforms provide access to both traffic-derived features and controller/host telemetry in a consistent time aligned manner. Therefore, this thesis will treat multi-plane observability as an important roadmap requirement and will investigate lightweight integration mechanisms that can enhance the robustness of anomaly detection without creating brittle coupling between planes.

Finally, SDN-based intrusion detection systems are inherently subject to real-time requirements. An IDS system must not only accurately detect anomalies in network traffic, but also do so in a timely fashion and with respect to the finite resources available to the system, particularly when the system is collocated with controller components that may already be under significant load during periods of adverse event activity.

These considerations lead to the development of streaming-compatible formulations of anomaly detection algorithms, including the use of sliding window representations of telemetry streams and the explicit consideration of latency, throughput, and computational footprint as primary evaluation metrics, and not simply as secondary implementation characteristics.

## 1.2 Scope and contributions

Within this motivation, the thesis advances a coherent path from evidence to methodology. It starts by analyzing the datasets that serve as the basis for SDN IDS research and identifying the empirical shortcomings that limit meaningful comparison and deployment-oriented evaluation. This view is operationalized in Chapter 3, which compares SDN IDS datasets and provides a controlled baseline on a representative benchmark; it produces not just an overview of the strengths and weaknesses of the available datasets but also a set of specific design requirements to convert survey level findings into action items for developing and testing SDN IDS pipeline.

With these requirements as guidance, Chapter 4 develops KRONOS-SDN, a realistic and reproducible benchmark specifically tailored to SDN intrusion detection under temporal and operational constraints. The dataset is built to provide multi-day continuity, contain a variety of different benign workloads, include a large number of different attack campaigns and offer multiple dimensions of observation of the network through a combination of network derived features and controller derived telemetry. Beyond dataset description, the chapter outlines an analysis process that allows for the use of the dataset for both benchmarking purposes and to provide methodological insights, including statistical characterization and supervised baselines that highlight common issues such as feature redundancy, the importance of temporal and idle time descriptors and the effects of class imbalance.

Using the methodological frameworks introduced in Chapter 5, the thesis then moves from supervised baselines to deep anomaly detec-

tion as a practical approach to SDN intrusion detection when labeled attack data are scarce or unavailable. This setting is particularly relevant when temporal patterns carry discriminative information, that is to say when attacks are better characterized by the evolution of traffic over consecutive time steps than by isolated flow records alone. Examples include persistent scanning activity, progressive increases in traffic volume, repeated bursts, changes in inter-arrival times and variations in active or idle periods. Chapter 6 therefore outlines a window-based pipeline on KRONOS-SDN, presents a comparative study of several well-known deep reconstruction-based anomaly detectors trained and tested with homogeneous training, calibration and scoring semantics, and evaluates the detectors based on their ability to detect anomalies, produce alerts quickly enough to prevent damage in terms of alert latency and operate efficiently enough to avoid affecting system performance in terms of throughput and computational footprint. Importantly, the evaluation reflects the need for deployable systems, namely the trade-offs required to make IDS models feasible in a real-world SDN environment.

The thesis also examines the potential for cross-plane integration through the development of a simple fusion primitive, i.e., rather than replacing the traffic-based detector with a single multi-source model, controller-side telemetry is used to estimate a signal that indicates the current operating regime and dynamically adjusts the sensitivity of the detector. The goal of this design is to lower the rate of false positives caused by normal workload variations while improving responsiveness to suspicious conditions occurring on the controller side, providing a reasonable compromise between static-threshold anomaly detection and dynamic decision-making.

Throughout the above steps, the thesis uses post-hoc interpretability as a methodological safeguard and as an operational tool. Feature attribution methods are used to verify the significance of the features that were identified through learning, to identify redundant features and to determine if the decision made by the detector is based on a fragile correlation or on a collection of correlated signals. These two levels

of interpretability link the dataset-centric assessments in Chapter 3 and Chapter 4 to the anomaly-detection evidence in Chapter 6 and support a consistent story in which empirical realism and methodological clarity mutually reinforce one another.

## 1.3 Thesis organization

The thesis follows an incremental progression from architectural fundamentals to empirical evidence and ultimately to deployment focused detection methodologies. Chapter 2 addresses foundational aspects of SDNs and the related security implications; the mentioned chapter details how the controller-centric design of SDNs provides a centralization of monitoring but also creates new vulnerabilities that need to be considered when making assumptions about Intrusion Detection System (IDS). Next the thesis takes a data-driven approach to the SDN IDS literature baselines with Chapter 3 addressing the SDN IDS literature and motivating explicit requirements for datasets, and with Chapter 4 operationalizing the requirements for datasets through the presentation of KRONOS-SDN and the analysis which supports the methodological decisions made in later chapters.

Then, the thesis focuses on developing IDS methods using machine learning under realistic constraint conditions. Chapter 5 presents the background referred to machine-learning-based IDS and establishes sliding window deep anomaly detection as a streaming compatible paradigm. In addition Chapter 6 presents the controlled experimental evaluation of KRONOS-SDN focusing on the simultaneous interpretation of detection performance and efficiency and introducing the cross-plane threshold modulation as a lightweight strategy for achieving context aware detection.

Chapter 7 summarizes the findings and outlines the limitations of the work presented in the thesis and presents potential directions for future research, including more sophisticated cross-plane fusion strategies.

Additional material is provided in the appendices to support both interpretability and reproducibility. Appendix A reports a SHAP-based

post-hoc interpretability analysis for the DDoS attack scenario contained in InSDN dataset, including beeswarm explanations for the considered baseline classifiers, in order to complement the performance discussion of Chapter 3 with feature-attribution evidence. Appendix B documents the specifics of the attack campaign to complement the information contained in Chapter 4, while Appendix C extends the interpretability evidence on KRONOS-SDN by providing SHAP summary beeswarm plots for the other classifiers beyond the representative case discussed in Chapter 4. Finally, Appendix D documents the controller-side context signal used for context modulation argument presented in Chapter 6.

# Part I

# SDN fundamentals

# Chapter 2

# Background

Contemporary *Software-Defined Networking* (SDN) architectures are best understood by contrasting them with traditional IP networking, where forwarding devices are engineered as autonomous systems. In legacy networks, each router or switch locally implements packet forwarding, maintains the control-plane logic required to compute and update forwarding state and exposes management interfaces for configuration and monitoring. Historically, this design has provided great scalability and fault tolerance; however, it is typically associated with a high level of operational complexity: policies must be replicated across many devices, reconfiguration can be slow and operators often lack a consistent, real-time global view of network-wide behaviour.

The limitations described above represent the reason why SDN proposes separating the Data Plane from the Control Plane, providing a logically centralized controller that governs the operation of all the forwarding nodes through programmable interfaces (D. Kreutz et al., 2014).

By decoupling control from forwarding, SDN enables faster policy updates, more agile traffic engineering and tighter integration of network management and security functions. At the same time, this architectural shift changes the security model: the controller and its supporting communication channels become high-value assets and their compromise or overload can affect the correctness and availability of the en-

tire infrastructure. As a result, intrusion detection and monitoring mechanisms are not merely an add-on capability but a core requirement to preserve the operational safety of SDN deployments, especially under adversarial conditions that target control logic, control-channel integrity, or flow-rule installation dynamics.

In consideration of the aforementioned aspects, the present chapter first revisits the functional division of legacy devices across the data, control and management planes; then uses this decomposition to introduce controller-based SDN architectures and the interfaces through which the controller communicates with network elements and higher-level applications. The concepts introduced here provide the necessary background for the remainder of the thesis, when discussing SDN-oriented intrusion detection datasets, the design and analysis of the proposed KRONOS-SDN dataset and the methodological choices adopted for the experimental evaluation. In particular, the foundations established in the current chapter will be leveraged in Chapters 5 and 6, where they support the rationale for the selection of telemetry sources and features and the interpretation of the empirical results within an SDN security context.

## 2.1 Functional Planes in Traditional Network Devices

Routers and switches perform a variety of tasks that can be grouped logically into functional planes. In particular, three planes are commonly distinguished: the *data plane*, the *control plane* and the *management plane*. Each plane encompasses a different class of operations and interacts with the others to provide end-to-end connectivity. Figure 1 provides a high-level representation of the traditional network architecture, where it is possible to notice the classical combination of forwarding and control logic within a single element.

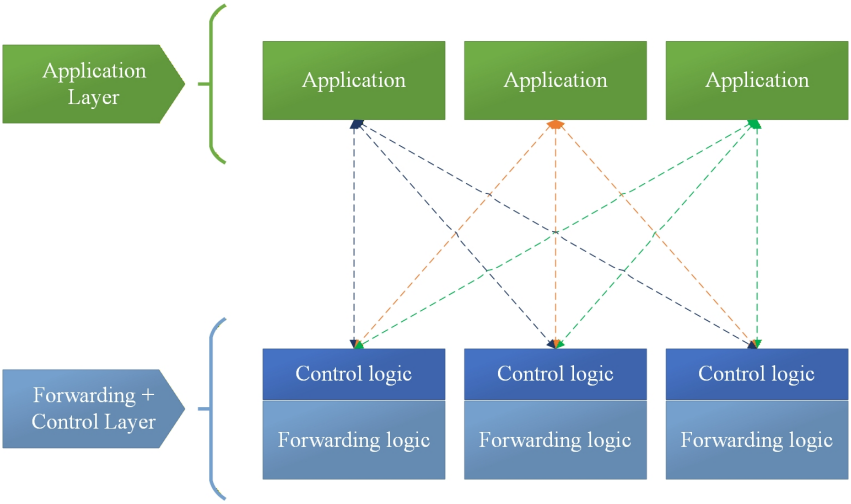



**Figure 1:** Logical architecture of a traditional network.

### 2.1.1 The Data Plane

The data plane, also called the *forwarding plane*, is responsible for processing individual traffic units, Ethernet frames, IP packets or more generic messages and deciding how they are forwarded. From the perspective of a router or switch, the data plane encompasses all the actions that are performed for each packet or frame when traffic arrives on an interface and is transmitted on one or more outgoing interfaces. Typical data plane operations include:

- Decapsulating and re-encapsulating IP packets into data-link frames when a router forwards traffic between different interfaces.
- Inserting or removing VLAN tags, for instance IEEE 802.1Q headers in switched Ethernet networks.
- Looking up a destination MAC address in a switch's forwarding table to select the appropriate output port.
- Matching the destination IP address of a packet against a router's IP routing table to determine the next hop.

- Applying security or policy rules, such as access control lists (ACLs) and discarding traffic that violates those rules.
- Transforming packet headers, for example during network address translation (NAT) or when building VPN tunnels and adding new encapsulation headers.

All such operations are executed at very high speed and must be performed for each individual message. Consequently, data plane processing must be extremely efficient, both in terms of latency and throughput, in order to sustain line-rate forwarding on modern high-speed interfaces.

### 2.1.2 The Control Plane

The data plane relies on local state, in particular forwarding tables, that must be computed and stored before the network element takes any forwarding decision. The *control plane* includes the protocols and algorithms responsible for constructing and updating this state. Control plane mechanisms populate data structures such as:

- The IP routing table, learned via dynamic routing protocols, static routes or directly connected networks.
- The Address Resolution Protocol (ARP) table in IPv4 or the Neighbor Discovery (ND) cache in IPv6.
- The MAC address table on Ethernet switches, built by observing source MAC addresses of incoming frames.
- The topology and active-interface set selected by the Spanning Tree Protocol (STP) to prevent Layer 2 forwarding loops.

In conventional networks, the mentioned functions are typically implemented in a *distributed* way. Each router is independently able to execute routing protocols such as OSPF; each switch runs STP and learns MAC addresses autonomously; each device maintains its own ARP or ND tables. Devices exchange control messages with their corresponding elements, collaboratively converging to a consistent view of the network

topology and reachability. The result is a distributed control plane in which decision logic and state are spread across all devices in the network.

### 2.1.3 The Management Plane

The *management plane* comprises the mechanisms that allow operators and management systems to configure, monitor and troubleshoot network devices. Unlike the control plane, which directly determines the forwarding behaviour by manipulating data plane state, the management plane focuses on supporting administrative tasks. Management Plane Protocols include the following functions:

- Remote access through mechanisms such as Telnet or SSH, allowing for configuration and/or monitoring.

- Simple Network Management Protocol (SNMP), used to query and modify management information bases (MIBs) and also to collect performance statistics from active network devices.

- Syslog and other mechanisms for log generation that export operational events and error messages to a remote collector.

A significant aspect of this model is the fact that a temporary unavailability of management-plane services does not necessarily prevent routers or switches from forwarding traffic, provided that the data plane remains operational and valid forwarding or routing state is already installed. However, such unavailability limits the ability to monitor, reconfigure, update or troubleshoot the devices. As an example, a router with no operational ssh service may still be able to run ospf and forward ip packets. The management plane is therefore crucial: it does not directly process user traffic but provides visibility and control over the other planes.

## 2.2 Limitations of the Traditional Distributed Control Plane

Conventional network control planes are distributed and thus offer many advantages. The distributed nature of the control plane provides natural scalability, as each device is responsible for its portion of control logic and can therefore continue forwarding packets using locally available information, even when communication to a central management system is lost.

However, this architecture introduces notable challenges:

- **Configuration complexity:** policies must be translated into device-specific configurations and replicated consistently across many nodes. Ensuring that ACLs, quality of service (QoS) policies and routing constraints are applied uniformly is prone to error (Leivadeas et al., 2023; David et al., 2023).

- **Limited global visibility:** each router or switch has only a partial view of the network. Implementing sophisticated traffic-engineering or security policies that depend on global state becomes impractical (Abderrahmane et al., 2024).

- **Slow innovation cycle:** introducing new behaviours often requires changes to distributed protocols or firmware upgrades across the entire device fleet, a process that is operationally expensive and slow (David et al., 2023).

- **Tight coupling of planes:** the functions for managing data, controlling the devices and management functions are strongly coupled in each device; therefore, it is not easy to make modifications to one plane without affecting other planes (Abderrahmane et al., 2024).

The limitation detailed above prompted a research for new architectures that decouple the planes and place the control logic centrally to allow for a more programmable and flexible network.

## 2.3 Controller-Based and Software-Defined Networking

Software-defined networking rethinks the distribution of the control plane by introducing a logically centralized entity, commonly referred to as the *controller*.

In an SDN architecture, the controller maintains a holistic view of the network and, to varying degrees, assumes responsibility for populating the forwarding state in network devices. While the forwarding hardware remains in the devices, the decision logic that determines *what* should be installed in those tables is partially or fully centralized.

Figure 2 shows a high-level graph of the SDN architecture with the separation of the three planes and the interface that interconnect them. Each component will be explained in the rest of the current section.

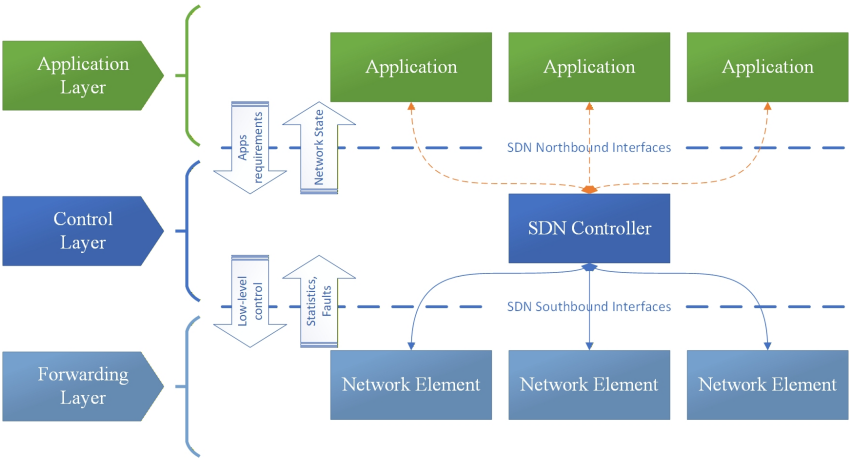



**Figure 2:** Logical architecture of a SDN network.

### 2.3.1 Centralized Versus Distributed Control

In traditional networks, protocols such as OSPF or STP run on each device and collaborate to reach a consistent routing or switching state. By

contrast, in a fully centralized SDN design the controller runs the decision logic and then transmits directly instructions to each device about which forwarding entry is to be installed. The devices themselves may execute minimal or no control plane protocols, as they primarily expose programmable forwarding tables to the controller.

Between the extremes of completely decentralized (distributed) control and completely centralized control, there exist many possibilities of design; consequently, SDNs can be deployed across an array of architectural options. Depending on the scenario, the primary function of the controller may be to act as a supervisory element: the controller will observe how distributed control plane protocols are operating, provide enhanced network wide visibility and allow for higher level automation and leave most aspects of traditional routing and switching protocols unchanged. However, in more radical designs, the controller can be configured to serve as the singular logical entity responsible for making routing and switching decisions; accordingly, forwarding devices are deliberately simplified so they can operate primarily as programmable data-plane components that implement the rules programmed by the controller.

Centralizing control has several potential benefits. It simplifies the development of complex policies, as the controller has access to global network state. It enables consistent network-wide configuration because a single entity computes and installs forwarding rules. It also facilitates rapid innovation: new applications can be developed on top of the controller without modifying the firmware on each device.

The advantages of the above-mentioned aspects of SDN are achieved at the cost of a new set of challenges (e.g., scalability, fault tolerance, the secure communication between the controller and the devices of the data plane and particularly, the security of the controller) that can be seen as the central part of SDN architecture. In case of deficiency in terms of controller security, the network would be characterized by a single point of failure. To overcome these problems, SDN architectures establish clear interfaces between the controller and the devices, as well as between the controller and external applications.

### 2.3.2 Southbound Interfaces

The *South Bound Interface* (SBI) is used to refer to the interface between the controller and the network device. The SBI allows the controller to configure the forwarding behaviour of the network devices and to receive information from them regarding their operational status, capabilities and any local events or changes.

A SBI typically consists of:

- A protocol that specifies message formats and semantics for exchanging information between controller and devices.
- An application programming interface (API) on both sides that exposes these capabilities to the controller's internal modules and to the software components running on the devices.

Different SDN solutions adopt different SBIs, reflecting diverse design goals. Notable examples include:

- **OpenFlow**[1], one of the earliest and most widely discussed SBIs, standardizes the model of a programmable switch and the messages used by a controller to install, update and delete flow entries in forwarding tables.
- **OpFlex**[2], used in Cisco Application Centric Infrastructure (ACI), which supports a policy-driven model where the controller expresses abstract intents and the devices perform local enforcement.
- **Traditional management protocols**, such as CLI over Telnet/SSH or SNMP, may also be used in a centralized architecture for devices without a native SDN protocol.

The structure of the SBI significantly impacts what a controller is able to accomplish. A solid SBI that has a high level of detail will enable

[1]Documentation available at: `https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf`.

[2]Documentation available at: `https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/kb/b_Cisco_ACI_and_OpFlex_Connectivity_for_Orchestrators.html`.

a controller to have granular control on forwarding behaviour, while a limited SBI will constrain how much the data plane can be controlled.

### 2.3.3 *The de-facto standard protocol:* OpenFlow

One of the earliest and most influential SDN architectures was defined by the Open Networking Foundation (ONF) under the paradigm of "Open SDN". In this model, the controller uses OpenFlow, the standard protocol mentioned in Subsection 2.3.2, as its primary southbound protocol to communicate with OpenFlow-capable switches.

OpenFlow introduces a standard abstraction of a switch that can act as a Layer 2 or Layer 3 device and support a flexible set of action rules in case of match. Each rule matches packet header fields, containing MAC addresses, VLAN identifiers, IP addresses and TCP/UDP ports and specifies actions such as forwarding to a port, modifying header fields or dropping the packet.

In the remainder of this thesis, the terms *flow entry* and *flow rule* are used interchangeably to denote a single OpenFlow table entry, i.e., a rule that matches packets on selected header fields and associates them with one or more actions.

The controller manages these flow tables by inserting, modifying and deleting entries as network conditions and policy requirements evolve. In this approach, most traditional distributed control plane functions are replaced by centralized logic in the controller and in the applications using its Northbound Interfaces, that will be introduced at Subsection 2.3.5.

The devices themselves become relatively simple forwarding elements that expose their capabilities to the controller via OpenFlow. This architecture emphasizes programmability and global optimization but also requires careful design of the controller platform to ensure resilience and scalability.

### 2.3.4 Workflow of OpenFlow-based Switching

As stated in Subsection 2.3.3, OpenFlow is the *de facto* standard southbound protocol that enables direct programmability of the data plane in

SDN environments (Open Networking Foundation, 2015; D. Kreutz et al., 2014).

An OpenFlow-enabled network switch is driven by either one or multiple flow tables and a control channel which enables communication between the switch and its remote controller. Flow tables are structured as sets of flow entries in decreasing order of their priorities, so that the most specific flow entries will be examined first. A flow entry describes the action to be taken on a group of packets and consists of a number of components including a number of match fields, a priority field, counter(s), a list of action(s) to be executed, a timeout parameter, a cookie defined by the controller and a set of flag values that define how the entry should be processed (Open Networking Foundation, 2015).

Match fields allow the switch to perform exact or wildcard matching over packet and port attributes, including the input port on which the packet arrived, header fields such as source and destination MAC or IP addresses, VLAN tags, transport-layer ports, as well as metadata carried over from previous tables in the pipeline.

When a packet reaches the switch, processing starts in the first flow table (table 0). The switch searches for all entries whose match fields are compatible with the packet and selects the one with the highest priority. Upon a successful match, the corresponding counters (e.g., number of packets and bytes) are updated and the actions associated with the entry are applied.

These actions define how the packet is handled: they may instruct the switch to forward the packet to one or more output ports, send it to the controller, drop it, modify selected header fields or direct it to a subsequent flow table, thereby realizing a multi-table processing pipeline. If no entry matches, the behaviour is governed by the table-miss rule (typically a low-priority default entry), which commonly results in encapsulating the packet in a control message towards the controller so that it can decide how to treat this new flow and, if needed, install a dedicated flow entry for subsequent packets. The lifetime of a flow entry is controlled by time-out parameters: an *idle timeout* specifies the maximum period of inactivity after which the entry is removed, while a *hard timeout* defines

an absolute upper bound on its duration regardless of use.

Cookies provide opaque identifiers that allow the controller to correlate flow entries with higher-level policies or applications and flags indicate special handling requirements or constraints (for example, whether the entry may be removed by certain operations) (Open Networking Foundation, 2015). Through OpenFlow control messages (such as `flow_mod` and `packet_in`), the controller continuously installs, updates and deletes flow entries over the secure channel, effectively steering the forwarding behaviour of the switch at the granularity of flows while keeping the actual packet processing in the data plane (D. Kreutz et al., 2014).

The preceding discussion has focused on the southbound direction of SDN communications, namely the interaction between the controller and the data-plane devices. In this model, the SBI will be responsible for defining *how* the forwarding behaviour is enforced on each device and how local state and events are sent back to the controller. However, the controller itself is rarely the end user of this information. Instead, it acts as an intermediary system that collects network state, applies control logic and exposes higher-level abstractions to external software components. To enable the controllers to act in this manner, SDN controllers provide *northbound interfaces* which allow orchestration systems, security applications and other network-aware services to query, monitor and reconfigure the network without interacting directly with the underlying devices. The next section introduces these northbound interfaces and details their role in providing network programmability.

### 2.3.5 Northbound Interfaces and Network Programmability

While southbound interfaces connect the controller to the network devices, *northbound interfaces* (NBIs) expose the controller's capabilities and the collected network state to external applications. A NBI allows higher-level software, that is to say: orchestration systems, security applications or custom-developed tools which are used to query the controller, react

to network events and request changes in network behaviour.

In practice, a NBI is realized as an API implemented by the controller software. Applications can run on the same physical or virtual host of the controller and link against native APIs or they can be deployed on remote hosts and communicate with the controller over the network. A widely used paradigm for remote NBIs is the *RESTful* API. In this model:

- Applications send HTTP requests to controller-defined uniform resource identifiers (URIs).
- The controller responds with structured data encoded in formats such as JSON or XML, which applications can easily parse and process.

By abstracting low-level device details behind a NBI, the controller enables rapid development of network-aware applications using mainstream programming languages and web technologies. This separation of concerns is a central principle of software-defined architectures.

## 2.4 From Traditional to Software-Defined Networks: A Comparative View

The transition from traditional distributed networks to software-defined, controller-based architectures can be framed as a shift in how the three planes are implemented and interact. This evolution does not necessarily imply that all traditional mechanisms are discarded. Many deployments adopt hybrid models where legacy protocols coexist with controller-based control or where different SDN technologies are applied in different network domains.

Nevertheless, the conceptual decoupling of planes and the introduction of programmable interfaces remain central contributions of SDN. They provide the foundation for advanced use cases such as intent-based networking, automated policy deployment, fine-grained traffic engineering and dynamic security enforcement. Table 1 details the crucial differences between traditional and SDN architectures.

**Table 1:** Comparison between conventional devices and SDN architectures.

| Plane | Conventional devices | SDN architecture |
|---|---|---|
| **Data plane** | Hardware-bound, local fast-path forwarding. | Hardware forwarding exposed as controller-programmable resources. |
| **Control plane** | Fully distributed: each device runs routing/switching logic. | Logically centralized in the controller. |
| **Management plane** | Per-device configuration and monitoring by operators. | Controller APIs enable network-wide orchestration. |

## 2.5 Specific threats in SDN landscape

SDN provides an environment with a very different threat profile than traditional IP-based networking systems due to foundational architectural aspects of the system that have been discussed so far. First, as mentioned in the previous sections, the control logic is concentrated in the controller, meaning that the component becomes a high-value target, because either the compromise or overloading of the controller may negatively affect the performance and security of the entire network. Second, SDNs also enable programmable routing of packets, this aspect allows for faster network policies to be implemented. However, SDN concepts open a door for attackers to take advantage of incorrect configurations, malicious SDN applications or unintended interactions in the control logic to steer traffic or bypass security controls. Finally, the use of standardized control interfaces (southbound protocol and north bound API) to manage data plane elements in an SDN expands the surface area upon which an attacker can focus to inject, modify, or deny control messages to the data plane thereby modifying the overall forwarding tables for the network.

The flexibility of these capabilities allows for quick policy updates and for easy traffic engineering, but in this way, the risks associated with a compromise of the controller or of the southbound interface, or a corrupted switch state can have disproportionate impacts on the entire net-

work compared to the attackers' efforts. A useful framework for visualizing the SDN threat surface is through the planes and interfaces, that is to say: end hosts and edge devices, the data plane for switches and flow tables, the control plane for the controller platform and its services, the application plane for the network apps and the channels/APIs between them (Deb et al., 2022; Melis et al., 2023).

### 2.5.1 Threats originating from end hosts and edge devices

In many operational SDN deployments, the attacker's most accessible foothold is a compromised endpoint or an untrusted subnet with a device attached to an edge switch (Varadharajan et al., 2019). From this position, *traffic-engineered* attacks can be launched without requiring privileged access to internal SDN components.

A prominent class is *flooding and rule-exhaustion*: by generating large volumes of packets or a high rate of new, distinct flows, an adversary can force repeated table misses and trigger excessive rule installations, especially in reactive forwarding, leading to switch resource exhaustion and denial of service (DoS). The SDN controller may also become a bottleneck if it must process an abnormal volume of `PACKET_IN` events (H. Wang et al., 2015).

A second category of attacks concerns *header manipulation and spoofing*. In these attack types, the attacker forges L2/L3 identifiers to bypass policies or to set up downstream attacks. Since SDN controllers rely on observed traffic and reported state from switches for topology discovery and policy enforcement, such modifications can have an amplified effect. Two examples of header manipulation and spoofing are: ARP cache poisoning (Hong et al., 2015) followed by an attack man-in-the-middle (MiTM) (Brooks et al., 2015), which allows an attacker to intercept and modify sensitive traffic flows.

The final category of attacks, *payload-based injection* attacks, occur when the packets sent to the controller contain sufficient information (metadata or payload) to support processing at the higher layers; an attacker can craft malicious packets to exploit vulnerabilities in applications and/or

controller side components (Deng et al., 2018) and thus elevate the original attack from an edge based attack to an attack on the control plane.

### 2.5.2 Data plane attacks against OpenFlow switches and forwarding state

OpenFlow-enabled switches, which serve as the network forwarding elements of the SDNs, are vulnerable to numerous high-impact attacks that result directly from its dual role of control-plane enforcement point for network policy and of sensor for network state. A critical family is *flow-table / rule flooding* and *Ternary Content Addressable Memory (TCAM) exhaustion*. Since flow tables and TCAM have finite capacity, an attacker can trigger an excessive number of distinct rules, preventing legitimate flows from being installed and thereby inducing service outages, degraded performance and unpredictable forwarding behaviour, including low-rate ("slow") exhaustion strategies (Pascoal et al., 2017). This is particularly damaging in reactive designs where the first packet of a flow is escalated to the controller and a new rule is installed on demand.

Far worse are integrity attacks on forwarding semantics through *fake flow-rule installation*. If an adversary can impersonate a controller, obtain privileged access on a switch, or exploit switch management interfaces, they may inject rules that mirror traffic, reroute flows through attacker-controlled nodes, drop selected traffic, or forward traffic in violation of policy (Yoon et al., 2017). These types of attacks could cause direct violations to confidentiality (traffic intercepted), integrity (traffic altered) and availability (selected or all drops). *Control message manipulation* aims to subvert the logical processing of the switch by forging or replaying OpenFlow commands, potentially causing widespread misconfiguration at line rate; more generally, compromised switches can exploit OpenFlow message semantics to trigger unsafe behaviours in SDN stacks (Jero et al., 2017).

Additionally, the data-plane may be compromised via *software and firmware vulnerabilities*. Embedded operating system running on switch platforms and the exposure of administrative utility functions create op-

portunities for buffer overflow attacks, malware inserted into OS/API components, privilege escalation, especially under weak management hardening (e.g., default credentials, exposed remote administration services). Hardware-rooted threats (e.g., firmware tampering and hardware trojans) represent a harder-to-execute but potentially disruptive class, as they undermine the trustworthiness of the forwarding device itself and can enable persistent, stealthy manipulation of network behaviour.

Two additional attack families are particularly relevant to SDN's reliance on telemetry: *false statistics reporting* and *timing-related attacks*. In the former, a compromised switch provides misleading counters, link status, or flow statistics to poison controller decisions (routing, load balancing, anomaly detection), potentially causing suboptimal paths or deliberate mis-routing (Yu et al., 2024). In the latter, attackers exploit semantic or measurement delays introduced by reactive control to infer policy and state: by probing and observing response-time differences between rule hits and misses (S. Liu et al., 2017), adversaries can learn communication patterns, infer access-control behaviour and profile monitoring strategies.

### 2.5.3 Attacks on the controller-switch control channel and controller ecosystem

The controller and the control channel are both valuable target areas due to their status as points of disruptions for the entire network. A malicious entity could potentially intercept, replay, or forge control messages through an unprotected controller-switch channel to affect how switches operate. In addition, compromising the controller-switch channel can rapidly lead to corruption of policy across the network, by causing the adversary to cause a false installation of rules, or suppression of legitimate rule update.

In addition to the portion of the attack surface consisting of the channel itself, the controller platform can also be attacked by manipulating the northbound APIs used by the controller, by installing malicious SDN applications, by exhausting resources (e.g., CPU), or by exploiting clas-

sical software vulnerabilities. Because controller services often maintain global topology and policy repositories, successful attacks can have systemic effects: incorrect topology views, broken routing, failed authentication services and impaired security controls that rely on centralized visibility. Application-layer threats and vulnerabilities at the northbound interface have been analyzed systematically in (Rauf et al., 2021), while controller-side containment and resilience against faulty/malicious apps are exemplified by (Shin et al., 2014).

### 2.5.4 Advanced and virtualization-related attacks

In modern implementations of SDN that use virtual switching and are managed by a cloud fabric, lateral propagation of attacks initiated in the data-plane is typical: attacks on virtual switch vulnerabilities or those of the data-path module provide an attacker with direct access to the host operating system (OS), as well as the control plane, thereby providing the attacker the manipulation capability of the traffic steering policies and/or controller instances located within the same infrastructure. Also, web-management interfaces provide a traditional web application attack surface (i.e., injection attacks, scripting attacks) that may serve as an indirect method for attackers to gain access to privileged configurations.

### 2.5.5 Attacks' impact in SDN

While many primitives (spoofing, flooding, MiTM) predate SDN, their *impact amplification* in SDN stems from the tight coupling between observed traffic, controller decisions and switch enforcement. Attacks that exhaust flow tables/TCAM disrupt the mechanism by which forwarding state is realized; attacks that poison statistics distort the control loop and attacks that compromise the controller or its channel can rewrite network-wide behaviour at scale.

Consequently, the most operationally damaging SDN threats tend to align with:

- *availability attacks* targeting controller responsiveness and switch memory such as DoS and rule exhaustion;

- *integrity attacks* manipulating flow rules and topology/telemetry, through rule injection, false reporting;
- *confidentiality attacks* that exploit programmable forwarding to enable persistent interception (MiTM, traffic mirroring).

Table 2 details the specific threats against SDN, what is the security target in terms of Confidentiality, Integrity and Availability (CIA) and the physical network element that is targeted.

## 2.6 Final Remarks

The classical architecture of routers and switches, built around distributed control and device-centric management, has served as the backbone of IP networks for decades. By explicitly distinguishing data, control and management planes, it is possible to understand both the strengths and the inherent limitations of this model, particularly in environments that demand rapid reconfiguration, fine-grained policy enforcement and tight integration with higher-level services.

Software-defined networking and controller-based architectures address these limitations by centralizing control logic in a controller, exposing programmable interfaces towards both devices (southbound) and applications (northbound) and retaining high-performance hardware data planes in the network devices. The resulting paradigm enables more flexible, automated and globally consistent network operation, while still leveraging the proven capabilities of ASIC-based forwarding hardware.

While SDN architecture has many benefits as detailed in Section 2.5, it introduces some security risks as well: the logical centralization and standardized control interfaces and planes of SDNs can lead to single points of failure and increase the number of potential entry points for malicious activity, increasing the risk of state-exhaustion and control-plane overload attacks, thus targeting the availability of the entire network; rule and telemetry manipulation attacks undermine the integrity of a network; programmable forwarding creates opportunities for malicious actors to intercept traffic. As a consequence, all of these compo-

nents, as well as the channels must be secured via hardened controllers-applications, hardened switches-controller communications and safeguards against state-exhaustion and misleading telemetry.

**Table 2:** Condensed mapping between representative SDN attack families, impacted security properties (CIA) and primary targets across SDN planes and components. **C**: Confidentiality, **I**: Integrity, **A**: Availability.

| **Attack family** | **CIA impact** | **Primary target** |
|---|---|---|
| Flooding & rule-exhaustion | A | Data plane, Control plane |
| Header manipulation & spoofing | I, C | Control plane services, Data plane policy enforcement |
| ARP poisoning / MiTM positioning | C, I | Data plane forwarding paths |
| Payload-based injection toward controller/apps | I, A | Application plane, Control plane |
| Flow-table / TCAM exhaustion | A | Data plane |
| Fake/malicious flow-rule installation | C, I, A | Data plane, Control plane |
| Control-message manipulation | I, A | Control channel (controller–switch), Data plane |
| Software/firmware vulnerabilities on switches | C, I, A | Data plane device |
| False statistics reporting / telemetry poisoning | I, A | Data plane telemetry, Control plane decision loop |
| Timing-related inference attacks | C | Control plane behaviour via Data plane observations |
| Northbound API abuse & malicious SDN applications | I, A | Application plane, Control plane |
| Web/management interface attacks | I, A | Application plane, Controller |

# Part II

# Datasets for Machine-Learning-Based Intrusion Detection Systems in Software-Defined Networks

# Chapter 3

# Comparative Study of SDN Intrusion Detection Datasets

Bridging the SDN background discussed in Chapter 2 with the empirical application field of this work, a central methodological implication emerges: in SDN, the intrusion-detection problem is shaped not only by packet-level artifacts on the data plane, but also by the programmability and feedback loops that tie together the control and data planes. In practical deployments, the controller's global view, the polling granularity of flow statistics and the timing of mitigation actions jointly determine what evidence is observable and at which temporal resolution, thereby influencing both the feature space available to learning algorithms and the threat models that can be realistically captured.

The result is that evaluating ML/DL-based IDS solutions in an SDN context cannot be separated from the datasets used to train and validate them. Since the traces collected by ML/DL-based IDS solutions in SDNs may not accurately represent SDN-related characteristics like the impact of flow rule installations, the load of the control plane or the near-real-time telemetry data, the performance reported by ML/DL-based IDS solutions may be based upon artifacts of the datasets rather than a fea-

sible/usable detection capability, as stated in (A. A. Bahashwan et al., 2023; Khalid et al., 2024).

This drives the data first approach taken in Chapter 3 to evaluate IDS pipelines prior to their development and evaluation: before IDS pipelines can be proposed and evaluated, the current body of evidence regarding realistic SDN intrusion datasets need to be reviewed with regards to aspects such as realism, documentation, temporal support and the threat profile.

The current chapter reviews the state of the art in IDS datasets for SDN, using a comparative analysis of SDN-oriented IDS datasets as part of Section 3.1; this comparative analysis will use a set of related dimensions to provide a comparison of the experimental realism, feature richness, reproducibility and the degree of experimental support for the temporal/near-real-time evaluations of the security relevance of each SDN-oriented IDS dataset. Based on the cross-dataset comparative perspective of the previous section, the comparative evaluation of the current chapter will narrow down its evaluation to InSDN, which, at the time of writing, represents the most commonly studied and publicly available IDS dataset collected within an SDN environment (Elsayed et al., 2020).

Then, Section 3.2 performs a baseline analysis of InSDN to both establish a controlled benchmark for classical supervised classifiers together with assessing feature-importance based on SHapley Additive exPlanations (SHAP) and critically identify structural limitations that constrain the dataset's value as a sole reference benchmark. The resulting empirical observations are synthesized into a set of concrete design requirements (DRs) that motivate the need for a new dataset capable of supporting more realistic, temporally rich and cross-plane evaluations. These requirements directly guide the design and generation of KRONOS-SDN, which is presented in Chapter 4.

## 3.1 Analysis of SDN-based intrusion detection datasets

Intrusion detection in SDN has initially relied on conventional network intrusion datasets such as KDD'99 (Stolfo et al., 1999), NSL-KDD (Tavallaee et al., 2009), UNSW-NB15 (Moustafa et al., 2015), CIC-IDS2017 (Sharafaldin et al., 2018a) or CSE-CIC-IDS2018 (*A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018)* 2018). While these datasets are still widely used, they were not collected in SDN environments and hence do not expose SDN-specific features such as OpenFlow statistics, controller events or control–data plane interactions. This mismatch between the training data and the operational context can lead to overly optimistic results and poor generalization when models are deployed in real SDN environments (Naeen et al., 2025).

To mitigate this problem, a number of datasets have been explicitly generated in SDN testbeds and made available to the community. These datasets vary significantly in terms of network architectures, attack scenarios, traffic generation tools, feature sets and public accessibility. Some focus narrowly on volumetric Distributed Denial of Service (DDoS) attacks against the controller, whereas others include a broader mix of attack types, including application-layer DDoS, brute-force attacks, web exploitation and others. More recently, SDN has also been integrated into vertical domains such as the Internet of Things (IoT) and microgrids, motivating specialized SDN-based datasets for those environments.

This section provides a comparative analysis of SDN-based intrusion detection datasets that are collected in an SDN or SDN–IoT testbed. The analysis builds upon the comparative study of eight SDN DDoS/IDS datasets presented in (Di Gennaro et al., 2025) and expands it by incorporating more recent SDN datasets such as NITSDN and IoT-oriented SDN dataset. The goal is not only to catalog datasets but also to critically assess how well they support the design and evaluation of ML-based intrusion detection systems in SDN environments.

### 3.1.1 Dimensions for comparison

In this chapter the study systematically compares SDN-based IDS datasets using a set of related dimensions that capture the characteristics of the datasets' experimental applicability as well as their security relevance.

**Attack coverage and threat model.** One key dimension is the breadth of attacks and threats captured by each dataset, and the specific SDN components being attacked (data plane, controllers, end-hosts, application plane), and thus which attack families are included within the dataset. For instance, datasets that only capture saturation of the data plane are generally less useful for evaluating the full scope of SDN security because the controller(s) and their interfaces represent a critical component of the SDN architecture.

**DDoS protocols and rate variability.** For DDoS-focused benchmarking, it is also important to differentiate what protocols are targeted in each dataset and if the dataset captures both high-rate flooding attacks as well as lower-rate flooding attacks. Furthermore, since many current SDN implementations are vulnerable to both volumetric pressure and low-rate attacks, capturing slow-rate, mixed-rate, or hybrid attacks is particularly important.

**Network architecture complexity.** In addition to the previously described factors, another significant dimension for comparing datasets is the level of complexity of the underlying testbed. This includes the type of SDN controller used; the number and roles of Open vSwitch (OVS) instances; the architecture of the network topology; whether the testbed was composed of virtual machines (VMs) only, or a combination of VMs and physical devices; and the total number of host devices and subnets present. While architecturally more complex deployments may more closely approximate actual SDN deployments, the greater complexity of such deployments also increases the engineering effort required to create them and may make replication of these studies more difficult.

**Traffic realism and collection methodology.** Datasets differ in how benign and malicious traffic are produced and collected. Benign traffic may originate from real production networks, be generated synthetically in controlled testbeds or be replayed from recorded traces. The realism of user behaviour, the diversity of the application mix and the orchestration of attacks and tools directly affect how representative the resulting traces are with respect to production SDN environments.

**Feature design and richness.** The number and types of features available for use in modeling and analysis represents yet another comparative axis. For example, some datasets only provide either packet-level or flow-level features, while other datasets may provide additional SDN-specific counters, temporally aggregated statistics, or information-theoretic descriptors of the network activity. In general, the number and types of features provided determine which machine learning models can be effectively trained to recognize anomalies in the network, as well as which hypotheses about the timing, location, and duration of attacks (or benign events) can be tested.

**Quality of the labels, granularity and task supported.** Datasets vary in terms of the degree of supervision they provide to researchers who wish to develop anomaly detection systems based upon the data. For example, some datasets allow researchers to train models to perform only a simple binary classification task (i.e., to classify packets/flows as either benign or malicious); while others allow researchers to train models to perform multi-class classification (i.e., to identify the family of attacks that a given packet/flow represents); and still others provide even more fine-grained labels (e.g., per-flow, per-window or per-burst labels). As researchers evaluate different approaches to developing anomaly detection systems, the quality of the labels, the severity of the class imbalance, and the rarity of certain classes of attacks all impact the validity of the results obtained.

**Public availability and documentation.** As a methodological matter, the availability of public data, as well as the quality of the documentation accompanying such data, greatly affects the scientific value of the data. For instance, some SDN datasets are publicly available in their entirety, providing researchers with access to the raw data, the scripts that generate the data, and detailed documentation about how to use the data. Other SDN datasets are privately owned, and/or only partially accessible. Publicly available data provides the foundation for replication, as well as for conducting fair, across-paper comparisons of different IDS pipelines.

**Support for temporal and real-time analysis.** Finally, the extent to which a dataset enables temporal and near-real-time evaluation is a distinct dimension. If a dataset exposes accurate time stamps, inter-arrival times, and meaningful state transitions, then it allows researchers to evaluate the performance of models that operate in near-real-time (online detection models) as well as those that operate after some delay (early detection models). Since delays in detecting anomalies in an SDN network can lead to delays in mitigating attacks, which can result in rapid overloading of the SDN controller(s), or rapid degradation of services, this characteristic is critical for many SDN deployment scenarios.

These comparative axes will be used to discuss and summarize the strengths and weaknesses of each individual dataset throughout the remainder of this section.

### 3.1.2 Early SDN intrusion datasets and baseline benchmarks

#### NSJ dataset (Niyaz et al.)

The NSJ dataset proposed by Niyaz *et al.* focuses on DDoS detection in SDN using deep learning (Niyaz, W. Sun, and A. Y. Javaid, 2017). The authors collect benign traffic from a real home wireless network over three days and malicious traffic from a separate laboratory environment. Malicious flows include seven different DDoS variants based on TCP,

UDP and ICMP flooding against an SDN controller.

The SDN testbed is built around a POX[1]controller, OpenFlow switches and virtual machines hosted on VMware ESXi. Packet captures are obtained using `tcpdump`[2], manipulated with `bit-twist`[3] and replayed with `tcpreplay`[4]. A total of 68 features are extracted from TCP/UDP/ ICMP traffic, including counts, flags and simple statistics and are fed to a stacked autoencoder used both for dimensionality reduction and classification.

From a comparative standpoint, NSJ offers the following strengths:

- it combines real benign traffic with lab-simulated attacks;
- it targets multiple DDoS variants;
- it exposes a relatively rich feature space tailored to SDN traffic.

However, the dataset is not publicly released and primarily concentrates on flooding attacks, without exposing controller state or broader attack types. This limits its role as a reusable benchmark and its coverage of the SDN threat landscape.[5]

**ZCA dataset (Zerbini et al.)**

Zerbini *et al.* construct an SDN dataset to evaluate anomaly detection and mitigation in SDN (Zerbini et al., 2019). The testbed uses a POX controller, Open vSwitch and a tree topology consisting of a root switch and four subnets with twenty hosts each. Malicious traffic comprises various DDoS attacks (TCP SYN, UDP and ICMP floods), as well as application-layer attacks generated with tools such as Hping3, LOIC, Metasploit, Hydra and SQLMap.

[1]Documentation available at: `https://github.com/noxrepo/pox`.

[2]Documentation available at: `https://github.com/the-tcpdump-group/tcpdump`.

[3]Documentation available at: `https://bittwist.sourceforge.io/`.

[4]Documentation available at: `https://tcpreplay.appneta.com/`

[5]The internal identifiers “NSJ”, “ZCA” and so on are used here only as compact labels, they do not appear in the original publications.

Traffic is captured in a Mininet[6]-based virtual environment over two days: one day devoted to normal traffic and the second day to mixed benign and malicious traffic. Six aggregate features are computed at one-second intervals, including bits/s, packets/s and entropy measures for source and destination IPs and ports. These statistics support the evaluation of ML models including Random Forests, SVMs and neural networks.

The ZCA dataset is publicly available and provides a clean environment with both volumetric and application-layer attacks, making it suitable for studying entropy-based detection methods. Its main limitations are the small feature set (only six high-level statistics) and a topology that, while non-trivial, remains relatively simple. It does not expose flow-level or controller-level features that would allow more detailed SDN-aware analyses.

**InSDN dataset (Elsayed et al.)**

InSDN is one of the most comprehensive and widely adopted intrusion datasets specifically designed for SDN (Elsayed et al., 2020). The testbed consists of four virtual machines: a Kali Linux attacker, an ONOS-based SDN controller, a Mininet emulator with Open vSwitch and a Metasploitable-2 host exposing vulnerable services. Traffic is captured with `tcpdump` and processed with CICFlowMeter [7], producing more than 80 features per flow, later reduced to a set of SDN-relevant metrics.

InSDN covers a broad range of threats: multiple DDoS modes (TCP SYN, UDP, ICMP floods and HTTP-based flooding using tools such as Slowloris[8] and Torshammer[9]), brute-force password guessing, web application exploits, botnet activities and attacks against services such as Samba. Normal traffic includes common services and realistic usage scenarios.

[6]Documentation available at: `https://mininet.org/`.

[7]Documentation available at: `https://github.com/ahlashkari/CICFlowMeter`.

[8]Documentation available at: `https://manpages.ubuntu.com/manpages/noble/man1/slowloris.1.html`

[9]Documentation available at: `https://github.com/lidarbtc/torshammer`

Because of its diversity of attack classes, rich feature space and public availability, InSDN has become a de facto benchmark for ML-based IDS in SDN (Ataa et al., 2024a; Mustafa et al., 2024). At the same time, it exhibits limitations that are common to many SDN datasets: the network architecture is quite simple and the size is modest. Moreover, some rare classes are underrepresented, making it harder to train robust multi-class classifiers.

#### NCLP dataset (Novaes et al.)

Novaes *et al.* (Novaes et al., 2020) propose a dataset for anomaly detection and mitigation in SDN using Long Short-Term Memory (LSTM) and fuzzy logic. The testbed uses a Floodlight[10] controller and a star topology with six switches and 120 hosts in Mininet. The dataset focuses on DDoS and port scanning attacks. DDoS scenarios include UDP, SYN, TFTP, DNS and NTP floods, as well as application-layer WebDDoS and Apache remote memory exhaustion.

Traffic is collected over two days: one day of normal traffic and one day of mixed malicious and benign traffic. Six features are extracted at one-second intervals: bits per second, packets per second and entropy metrics for source and destination IPs and ports. The dataset is publicly accessible and has been evaluated with a variety of classifiers, including a hybrid LSTM–fuzzy logic model that outperforms more conventional methods on the provided features.

Compared to InSDN, NCLP emphasises temporal statistics and DDoS rate variability rather than feature richness. It is well suited for time-series and entropy-based methods, but the limited feature set and focus on DDoS/port scanning limit its usefulness for more general SDN intrusion detection tasks.

[10]Documentation available at: `https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview`.

**SDN-SlowRate-DDoS dataset (Yungaicela et al.)**

Yungaicela-Naula *et al.* (Yungaicela-Naula et al., 2023) introduce the SDN-SlowRate-DDoS dataset to study application-layer DDoS attacks in a realistic SDN data center. The testbed is a hybrid environment with real hardware (HP Aruba and NEC switches) arranged in a spine–leaf topology and controlled by ONOS. Traffic is captured using CICFlowMeter, enabling both packet-level and flow-level statistics.

The dataset focuses on slow rate DDoS attacks on application layer designed to deplete server resources while remaining under traditional detection thresholds. Legitimate traffic includes FTP and video streaming, allowing the authors to evaluate false positives and assess the impact of slow-rate attacks on user-facing services. Thirteen flow features are exported and used to train a LSTM-based detector.

The main strength of this dataset is its emphasis on slow-rate, application layer DDoS in a physical SDN environment, which complements the primarily virtual and high-rate DDoS focus of earlier datasets. However, it concentrates on a single attack family and does not include high-rate volumetric DDoS or other intrusion classes, which constrains its role as a general-purpose SDN benchmark.

**AAHHBA dataset (Aladaileh et al.)**

Aladaileh *et al.* (Aladaileh et al., 2022) present an SDN dataset centered on detecting DDoS attacks against the controller using Renyi joint entropy and dynamic thresholds. The testbed is implemented in Mininet, with a POX controller managing a linear topology of 64 hosts and OpenFlow switches. The dataset models both low-rate (5 packets/s) and high-rate (33 packets/s) DDoS traffic, generated by one or multiple attackers targeting one or more victims.

Traffic is sampled every five seconds over 60-minute simulations across eight attack conditions. Seven features are recorded, primarily IP address statistics used to compute joint entropy values. While the dataset is innovative in its explicit use of mixed-rate DDoS scenarios and entropy-based analysis, it remains limited to DDoS attacks and a small

set of features. Moreover, the dataset is not openly released, restricting its adoption as a community benchmark.

**ASMK dataset (Ahuja et al.)**

Ahuja *et al.* propose an automated ML-based framework for DDoS detection in SDN, which implicitly defines the ASMK dataset (Ahuja et al., 2020). Their Mininet-based testbed uses a tree topology with OpenFlow switches and a single SDN controller (Ryu or POX, depending on implementation variants). Normal and malicious traffic are generated using tools such as Hping3 and captured via Wireshark.

The dataset targets volumetric DDoS attacks (TCP SYN, UDP, ICMP floods) at different intensities. Twenty-three flow-level features are extracted, including packet sizes, flow durations and protocol-specific counters. The authors evaluate Decision Trees, Random Forests and neural networks, showing that Random Forests achieve excellent performance on the collected data.

In comparative terms, the ASMK dataset offers a richer feature space than AAHHBA and NCLP and its focus on controller-directed DDoS aligns closely with SDN threat models. However, like AAHHBA, it is not publicly available and is restricted to DoS/DDoS attacks, with no support for other intrusion classes.

**HLD-DDoSDN dataset (Bahashwan et al.)**

Bahashwan *et al.* introduce HLD-DDoSDN, a benchmark dataset specifically designed to study high- and low-rate DDoS attacks in SDN (A. Bahashwan et al., 2024). The testbed is a Mininet environment with a linear topology, 64 hosts, a single POX controller and an OpenFlow vSwitch. Attack traffic consists of TCP SYN, UDP and ICMP floods at both high (33.33 packets/s) and low (5 packets/s) rates; benign traffic is generated concurrently to emulate realistic load.

Packet captures are obtained with Wireshark[11] and CICFlowMeter is used to extract 71 features, including size distributions and inter-arrival

[11]Documentation available at: `https://www.wireshark.org/`.

time statistics. The dataset is publicly released and evaluated with deep multilayer perceptrons for both binary and multi-class classification, achieving high detection performance.

Relative to earlier DDoS datasets, HLD-DDoSDN stands out for its combination of high/low-rate attacks, rich feature set and public availability. At the same time, it still covers only flooding attacks and operates in a simple environment, leaving out other important SDN threats, architecture complexity and device heterogeneity.

### 3.1.3 Recent SDN intrusion datasets for vertical domains

#### NITSDN dataset

Khanal *et al.* (Khanal et al., 2023) propose NITSDN, an SDN dataset explicitly designed for ML-based IDS development. The dataset is collected in an emulated SDN environment where network traffic is captured using Wireshark and converted into flow-level records. The authors emphasise the need for SDN-specific features and report that IDS models trained on NITSDN can reach high accuracy (above 99%) when using decision-tree classifiers.

NITSDN is motivated by limitations observed in InSDN, in particular the small number of botnet instances. It introduces additional attack classes and a larger number of malicious samples to better support ML training. NITSDN has been reused in subsequent work on real-time anomaly detection frameworks in SDN, often in combination with InSDN.

From a comparative perspective, NITSDN complements InSDN by increasing the volume of malicious samples and emphasizing SDN- specific traffic features. However, the limited available documentation suggests that the testbed remains limited, due to no exposure of controller to possible attacks or host-level metrics characterization.

#### IoT-oriented SDN datasets

Beyond generic SDN networks, SDN has been combined with IoT and cyber-physical systems, leading to several SDN–IoT intrusion datasets.

Two representative examples are the SDN-IoT dataset (Sarica et al., 2020) and the more recent IoT-SDN-DH dataset by Dhirar and Hamad (Dhirar et al., 2025).

The SDN-IoT dataset was introduced as a benchmark for intrusion detection in IoT environments managed through SDN. It includes two traffic collections modeling static and dynamic IoT scenarios, with approximately 27.9 and 30.2 million records, respectively. The dataset contains benign traffic and five attack categories, namely DoS, DDoS, port scanning, OS fingerprinting and fuzzing, and provides 33 SDN-retrievable flow-level features collected through an SDN application querying switch flow entries at regular intervals.

More recently, Dhirar and Hamad presented the "Internet of Things Software-Defined Network Intrusion Detection Dataset: Inter- and Intra-Domain" (IoT-SDN-DH). This open-access dataset is built from simulated SDN-based IoT networks and contains approximately 40 million flow records. Eighty-six features are extracted to describe normal traffic and fifteen attack types across two flow profiles (inter- and intra-domain). The authors evaluate six ML/DL algorithms (including Decision Trees, Random Forests, deep neural networks, k-Nearest Neighbours, Bernoulli Naïve Bayes and logistic regression) and position their dataset as a comprehensive resource for anomaly-based IDS in IoT settings based on SDN architectures.

In parallel, other domain-specific SDN datasets have been proposed, such as SDN-MG25 for microgrid environments, which combines SDN traffic, system-call traces and power measurements to study attacks on SDN-based energy systems (Zhang et al., 2025). These datasets highlight the growing importance of SDN in verticals where network security has safety or mission-critical implications.

### 3.1.4 Cross-dataset comparison

Table 3 summaries key network-level characteristics of the main SDN-based IDS datasets discussed above, extending the comparison originally presented in (Di Gennaro et al., 2025) to include NITSDN and SDN–IoT

datasets. Datasets are characterized in terms of protocols, the presence of rate variability, controller, number of features and public availability.

**Table 3:** Comparison of SDN-based IDS datasets by network features, controller and availability.

| Dataset | TCP | UDP | ICMP | App. | Rate var. | Controller | No. Feat. | Avail. |
|---|---|---|---|---|---|---|---|---|
| NSJ | ✓ | ✓ | ✓ | – | – | POX | 68 | No |
| ZCA | ✓ | ✓ | ✓ | ✓ | limited | POX | 6 | Yes |
| InSDN | ✓ | ✓ | ✓ | ✓ | limited | ONOS | 83 | Yes |
| NCLP | ✓ | ✓ | – | ✓ | high/low | Floodlight | 6 | Yes |
| SDN-SlowRate-DDoS | – | – | – | ✓ | low-rate | ONOS | 13 | Yes* |
| AAHHBA | ✓ | ✓ | – | – | high/low | Floodlight | 7 | Yes |
| ASMK | ✓ | ✓ | ✓ | – | high/mod. | Ryu/POX | 23 | Yes |
| HLD-DDoSDN | ✓ | ✓ | ✓ | – | high/low | POX | 71 | Yes |
| NITSDN | ✓ | ✓ | ✓ | ✓ | – | Ryu | 82 | No |
| SDN-IoT | ✓ | ✓ | ✓ | ✓ | – | Ryu | 33 | Yes |
| IoT-SDN-DH | ✓ | ✓ | ✓ | ✓ | – | Ryu | 86 | Yes |

Several trends emerge from this comparison:

- **DDoS-centric focus.** Most datasets are either exclusively DDoS-centric (NSJ, AAHHBA, ASMK, HLD-DDoSDN) or treat DDoS as the main threat category, with port scanning and a few application-layer attacks as secondary classes (ZCA, NCLP). InSDN, NITSDN and SDN–IoT datasets broaden the attack spectrum by including brute-force, web exploits, botnets and service-specific attacks, but flooding attacks still dominate the class distribution.

- **Rate variability.** Only a subset of datasets explicitly model both high- and low-rate DDoS (NCLP, AAHHBA, HLD-DDoSDN) or slow-rate application-layer DDoS (SDN-SlowRate-DDoS). This is important because SDN controllers can be targeted by both aggressive floods and stealthy, low-and-slow patterns.

- **Feature richness vs. interpretability.** Datasets with many CICFlowMeter like features (InSDN, HLD-DDoSDN, IoT-SDN-DH) allow flexible feature selection and support sophisticated ML models, but also introduce higher dimensionality and redundancy. Conversely, datasets with only a few aggregated features (ZCA, NCLP,

AAHHBA) are easier to analyze and visualize, but may lack the granularity needed to capture subtle anomalies or to generalize across scenarios.

- **Public availability.** Only a subset of datasets is fully accessible to the community (ZCA, InSDN, NCLP, SDN-SlowRate-DDoS, HLD-DDoSDN, SDN–IoT, IoT-SDN-DH, NITSDN). Others (NSJ, AAHHBA, ASMK) remain private, which restricts their impact on reproducible research and makes cross-paper comparisons difficult.

Complementing Table 3, Table 4 compares datasets by architectural complexity, including the number of OVS instances, environment type and host count. Most datasets rely on Mininet-based virtual testbeds with a single controller and a consistent number of hosts (from 5 to about 120), while only SDN-SlowRate-DDoS uses a hybrid testbed with real switches. For NITSDN, the original work describes an SDN testbed emulated in Mininet with a Ryu controller, but it does not report a fixed number of Open vSwitch instances or end hosts; for this reason, these entries are marked as "n/a" in Table 4. Similarly, the SDN-IoT line aggregates two variants of the dataset introduced by Sarica and Angin, with 5 and 10 IoT devices respectively; additionally, when accounting for background, server and attacker hosts, the two variants result in 12 and 17 total hosts.

**Table 4:** Comparison of SDN-based IDS datasets by architecture.

| Dataset | OVS instances | Environment | No. Hosts | Hybrid / physical |
|---|---|---|---|---|
| NSJ | 1 | Virtual | 15 | No |
| ZCA | multiple | Virtual | 80 | No |
| InSDN | multiple | Virtual | 5 | No |
| NCLP | multiple | Virtual | 120 | No |
| SDN-SlowRate-DDoS | multiple | Hybrid | 20 | Yes |
| AAHHBA | 1 | Virtual | 64 | No |
| ASMK | multiple | Virtual | 13 | No |
| HLD-DDoSDN | 1 | Virtual | 64 | No |
| NITSDN | n/a | Virtual | n/a | No |
| SDN-IoT | 1 | Virtual | 12 / 17[a] | No |
| IoT-SDN-DH | 6 | Virtual | 8 | No |

The dominance of small to medium-size topologies implies that many datasets do not fully capture the heterogeneity, latency and specific behaviours of production SDN deployments.

### 3.1.5 Discussion and open challenges

As the ecosystem of SDN-based IDS datasets continues to grow, a number of significant advancements have taken place over the last few years. Initial studies, like those presented in NSJ and ZCA, were able to show that it was feasible to collect SDN-specific traffic for training ML models used in detecting controller-targeted Distributed Denial of Service (DDoS) attacks. Since then, subsequent datasets (like NCLP, AAHHBA, ASMK and HLD-DDoSDN) continued to evolve by addressing the challenges of capturing mixed rate DDoS attacks, entropy-based features and high-vs.-low rate attack regimes.

InSDN was a turning point because it provided a publicly accessible, rich dataset that included a wide variety of attack types, a large feature set and has since been widely reused in SDN IDS research. More recently, datasets like NITSDN, SDN-IoT, IoT-SDN and SDN-MG25 have begun to focus on specific verticals and expand attack coverage.

Although the ecosystem of SDN-based IDS datasets has made great strides in terms of advancement, a number of significant shortcomings continue to exist:

- **Narrow attack diversity.** Many datasets remain focused on volumetric DDoS attacks against the controller, with limited representation of other threats such as control-plane poisoning, topology manipulation, flow-rule tampering, lateral movement or advanced persistent threats. Only a few datasets (e.g., InSDN, IoT-SDN) include broader categories such as brute-force, web exploitation and botnets and even there some classes are underrepresented.

- **Synthetic and short-lived traffic.** Most datasets utilize either synthetic traffic or traffic generated from short-term simulations in Mininet or other similar network simulators. Although these meth-

ods simplify experimentation, they also neglect to capture long-term seasonal patterns, user behaviour dynamics and/or hardware artifacts that are commonly found in operational networks. As a result, models trained using these datasets may perform well when tested using the same data, but are unlikely to generalize when deployed into an actual operational network.

- **Limited device heterogeneity and multi-controller scenarios.** The majority of datasets utilized to date have used homogeneous virtual hosts and a single SDN controller. However, most actual deployments of SDNs consists of multiple devices with varying characteristics (e.g., servers, IoT nodes, etc.), and switches with various properties. Only a small number of datasets (i.e., SDN-SlowRate-DDoS and the SDN-IoT) have made attempts to create heterogeneous hardware or large IoT populations.

- **Temporal and real-time features.** Some datasets provide windowed statistics (e.g., NCLP and AAHHBA); however, many others do not report exact timestamps or inter-arrival times, nor do they provide access to controller event logs that would allow researchers to evaluate early-detection and streaming IDS algorithms. Without such features, researchers are unable to evaluate the performance of near-real-time detection strategies. These detection strategies are critical in SDN because controller overload can rapidly escalate if not addressed in a timely manner.

- **Reproducibility and tooling.** Even though many datasets are publicly available, the tools necessary to generate the traffic, configure the testbed, and extract features from the traffic are not always provided. This creates difficulties for researchers wishing to recreate the original experimental setup, regenerate the data under different conditions, or extend the dataset to support new attack scenarios.

Therefore, recent surveys on SDN intrusion detection have concluded that the community currently lacks large-scale, multi-attack, multi-plane

datasets that combine realistic SDN traffic with rich feature sets and public accessibility (A. A. Bahashwan et al., 2023; Khalid et al., 2024). The comparison performed in this section supports the following conclusion: although current SDN-based datasets are useful and have contributed to our knowledge of how to detect attacks in SDN, each of the datasets represents only a subset of the requirements for developing robust and generalizable machine learning-based intrusion detection systems.

Thus, from a thesis perspective, these shortcomings represent motivation for designing a new SDN dataset that:

- covers a broader range of attacks beyond DDoS;
- integrates multi-layer information (e.g., controller events, data-plane flows and host metrics);
- captures long-term traffic in more realistic, heterogeneous environments;
- is released with complete documentation and tooling.

The dataset and methodology developed in Chapter 4 are designed to address all of these shortcomings and develop a more representative benchmark for SDN intrusion detection.

## 3.2 Baseline analysis of the InSDN dataset

The methodological path adopted within this thesis concerning the comparative analysis of SDN-based intrusion detection dataset focuses on a systematic analysis of the InSDN dataset, which is, to the best of current knowledge, the most widely studied, publicly available, intrusion detection dataset specifically collected in a Software-Defined Networking (SDN) environment (Elsayed et al., 2020). The objective is twofold.

First, the aim is to use InSDN as a controlled benchmark to investigate how traditional machine learning classifiers behave when trained on SDN traffic and to understand which flow-level features are actually exploited during classification. Second, we critically assess the strengths

and possible gaps of InSDN as a reference dataset and derive a set of requirements that motivate the design of a new, more realistic dataset (KRONOS-SDN), which is presented in Chapter 4.

### 3.2.1 Overview of the InSDN testbed and traffic collection

InSDN was generated in a virtualized SDN testbed designed to emulate a simple yet representative network architecture (Elsayed et al., 2020). The environment consists of a Kali Linux virtual machine acting as the attacker, a Metasploitable2 virtual machine used as the victim for server-side attacks, an ONOS controller and an Open vSwitch (OVS) instance running on an Ubuntu-based virtual router. Additional Mininet hosts and bridges controlled by ONOS emulate, respectively, end systems and internal links. Within this testbed, the authors collected traffic flows under both benign and malicious conditions, exporting them in pcap and CSV format with flow-level statistics extracted from CICFlowMeter.

The resulting dataset comprises 343 939 bidirectional flows, of which 68 424 correspond to normal traffic and 275 515 to attack traffic. Traffic is categorized into TCP, UDP and ICMP flows and further organized into three logical subsets: purely normal traffic, attacks targeting the Metasploitable2 server and attacks targeting the OVS machine. For each flow, more than 80 statistical attributes are provided, such as duration, number of packets, number of bytes, inter-arrival times and various counters derived from the headers, excluding direct identifiers of the source and destination machines.

From an analytical standpoint, it is convenient to organize the extracted flow features into a small number of semantic groups (Di Gennaro et al., 2024). A first group comprises *network identifier* attributes, such as IP addresses, transport-layer ports and the protocol field, which primarily describe who is communicating and through which service. A second set captures *packet-based* and *byte-based* measurements, i.e., forward and backward packet counts and their corresponding byte volumes, summarizing the intensity and asymmetry of the exchange. Tem-

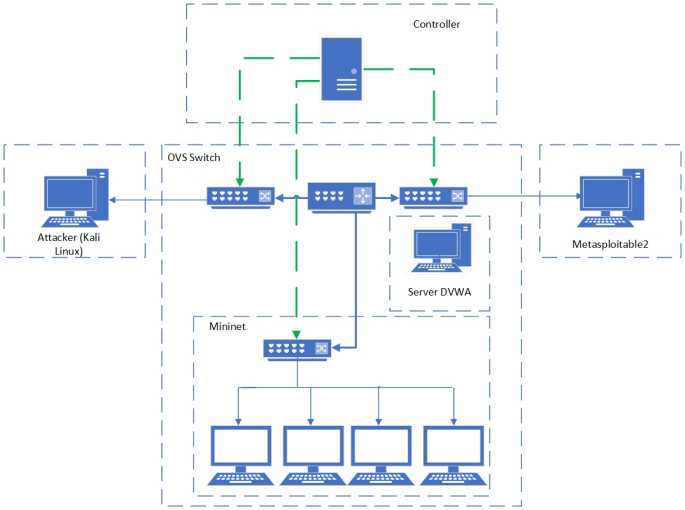



**Figure 3:** Network architecture used as testbed for InSDN dataset generation.

poral behaviour is represented through *inter-arrival time* features and *flow timers*, which encode the dynamics of packet spacing as well as active and idle periods within the flow. Additional descriptors are derived from protocol semantics, most notably *flag-related* counts (e.g., SYN, RST, PUSH), which reflect connection setup, teardown and anomalous signaling patterns. Finally, the feature set includes higher-level *flow descriptors* computed as statistics over packets and bytes within the flow (e.g., extrema and dispersion measures), together with *sub-flow descriptors* that characterize finer-grained segments of the same communication instance.

This organization provides a convenient conceptual framework for reasoning about which parts of the traffic behaviour are actually exploited by classifiers and which remain fundamentally unused.

Figure 3 shows the network architecture employed to generate both benign and malicious traffic.

### 3.2.2 Attack scenarios and class structure

From a security perspective InSDN focuses on a limited, even though relevant, set of attack typologies that specifically affect SDN infrastructures (Elsayed et al., 2020). The dataset includes the attack categories presented in Table 5.

**Table 5:** Attack categories considered in the InSDN dataset.

| Attack category | Description |
|---|---|
| Denial of Service (DoS) | Single-source resource exhaustion aimed at disrupting availability of the data and control plane. |
| Distributed DoS (DDoS) | Multi-source flooding that saturates bandwidth, state or controller processing capacity. |
| Brute-force attack (BFA) | Repeated authentication attempts to guess credentials and obtain unauthorized access from the Metasploitable2 server. |
| Exploitation (U2R) | Remote exploitation of vulnerable services to gain access on the target server hosting Metasploitable2. |

Alongside these malicious scenarios, the dataset contains normal traffic that mimics typical usage of Internet services (HTTP/HTTPS, SSH, mail, DNS and so on) directed to the Metasploitable2 server and mininet-based hosts. Within the used methodology, this setup is treated as a multiclass intrusion detection problem where each attack type constitutes a distinct class; however, the actual analysis is performed by training one-vs-rest binary classifiers per attack type, in line with the original InSDN study.

### 3.2.3 Preprocessing and experimental protocol

The first step of the analysis consists in consolidating the three CSV files provided with InSDN (corresponding to Metasploitable2, OVS and normal traffic) into a single dataframe and applying a uniform preprocessing pipeline. This pipeline includes the steps presented in Table 6.

Figure 4 shows the methodology adopted to analyze the InSDN dataset.

**Table 6:** Pre-processing and experimental protocol adopted for the binary classification tasks.

| Step | Description |
|---|---|
| Exploratory data analysis | Check missing values, outliers and low-variance attributes; apply mild filtering / simple transforms if needed. |
| Removal of socket and protocol identifiers | Drop `Timestamp`, `IP source/destination`, `Protocol`, `Source/Destination port`, `ID` to reduce leakage and memorization. |
| Binary problem formulation | Build one binary task per attack type (DoS, DDoS, BFA, U2R) vs. normal traffic. |
| Feature scaling and encoding | Scale continuous features; encode labels as 0 (normal) and 1 (malicious). |
| Cross-validation | Use 5-fold cross-validation for robust estimates with acceptable cost (Elsayed et al., 2020). |

On top of this shared preprocessing, the set of baseline machine learning models originally evaluated on InSDN was used for further deepening towards feature-selection studies: Decision Trees (DT), Random Forests (RF), Support Vector Machine (SVM), Naïve Bayes (NB), $k$-Nearest Neighbours (KNN) and Multilayer Perceptrons (MLPs) (Song et al., 2015; Parmar et al., 2019; Hearst et al., 1998; Rish, 2001; Guo et al., 2003; G. Singh et al., 2014). Hyperparameters follow standard configurations, including the specific MLP architecture with two hidden layers (100 and 80 neurons), sigmoid output and ReLU activations trained for 10 epochs with batch size 100 and learning rate 0.001 (Di Gennaro et al., 2024).

Performance is evaluated using confusion-matrix-based metrics (accuracy, precision, recall and $F_1$-score) averaged across five folds using stratified k-fold cross validation, as well as per-class scores where relevant. This makes it possible to quantify how well each model distinguishes normal flows from each attack type under controlled, reproducible conditions.

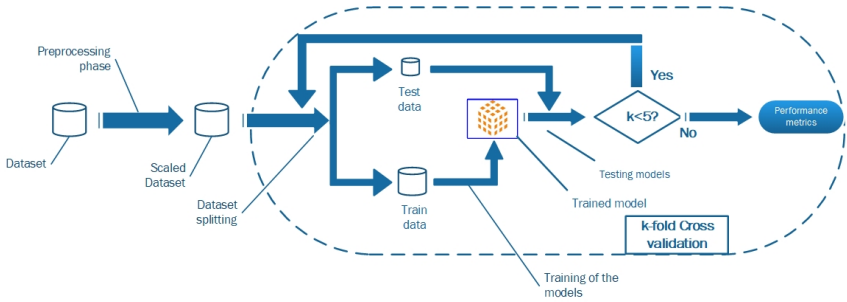



**Figure 4:** InSDN analysis methodology. Reprinted from Di Gennaro et al., 2024. © 2024 IEEE.

### 3.2.4 SHAP-based feature-importance analysis

The central methodological contribution of the preliminary study consists in coupling the above supervised learning pipeline with a feature-importance analysis based on SHapley Additive exPlanations (SHAP) (Marcílio et al., 2020). SHAP originates from cooperative game theory and assigns to each feature a Shapley value that quantifies its own contribution to the prediction, averaged over all possible subsets of features. When applied to intrusion detection, SHAP provides an interpretable link between low-level flow statistics and the decisions produced by a classifier.

For each attack type and learning algorithm, the pipeline consists of the steps detailed in the following paragraphs.

#### Full-feature training under stratified cross validation

The supervised learning task is formulated for all families of attacks as a binary classification problem (attack vs. benign) by selecting and combining from the dataset only those rows that relate to the appropriate class. All attributes related to socket and timestamp were removed, thus creating the *complete* set of features. Normalization of each attribute was accomplished using parameters calculated only on the training sets and

applied to the corresponding test set to prevent an information leak.

As such, k-fold cross-validation was used with $k = 5$ to evaluate classifier performance under a stratified condition and thus preserve the benign to attack ratio across all folds. Each classifier is trained on $k-1$ folds and tested against the $k^{th}$ fold, and for each test fold accuracy, precision, recall and $F_1$-score are computed based on the confusion matrix. These performance statistics serve as a *baseline* to compare subsequent results when performing the feature selection process for the purpose of obtaining interpretable models. The results of this part are provided in Table 7, where the values highlighted in bold indicate the best-performing results for each metric within each experimental run.

**Table 7:** Per-attack performance metrics (precision, recall, F1-score and accuracy) for the evaluated classifiers. Adapted from Di Gennaro et al., 2024. © 2024 IEEE.

| Attack | Metric | SVM | DT | NB | KNN | RF | MLP |
|---|---|---|---|---|---|---|---|
| DDoS | Pr | 0.99969 | 0.99993 | 0.99619 | 0.99993 | **1.00000** | 0.99997 |
| | Recall | 0.99967 | 0.99993 | 0.99643 | 0.99993 | **1.00000** | 0.99996 |
| | F1 | 0.99968 | 0.99993 | 0.99630 | 0.99993 | **1.00000** | 0.99997 |
| | Acc | 0.99968 | 0.99993 | 0.99630 | 0.99993 | **1.00000** | 0.99997 |
| DoS | Pr | 0.99147 | 0.99942 | 0.99712 | 0.99926 | **0.99981** | 0.99922 |
| | Recall | 0.99312 | 0.99950 | 0.99117 | 0.99933 | **0.99985** | 0.99921 |
| | F1 | 0.99224 | 0.99946 | 0.97118 | 0.99929 | **0.99983** | 0.99921 |
| | Acc | 0.99234 | 0.99947 | 0.97161 | 0.99930 | **0.99984** | 0.99922 |
| BFA | Pr | 0.99455 | 0.99461 | 0.56439 | 0.99642 | **0.99996** | 0.99617 |
| | Recall | 0.99455 | 0.99633 | 0.92905 | 0.99815 | **0.99822** | 0.98925 |
| | F1 | 0.99455 | 0.99547 | 0.57693 | 0.99728 | **0.99909** | 0.99269 |
| | Acc | 0.99457 | 0.99964 | 0.86438 | 0.99729 | **0.99993** | 0.99943 |
| U2R | Pr | **0.99996** | **0.99996** | **0.99996** | 0.83330 | **0.99996** | **0.99996** |
| | Recall | **0.83333** | **0.83333** | **0.83333** | 0.83330 | **0.83333** | **0.83333** |
| | F1 | **0.89998** | **0.89998** | **0.89998** | 0.83330 | **0.89998** | **0.89998** |
| | Acc | **0.99993** | **0.99993** | **0.99993** | 0.99985 | **0.99993** | **0.99993** |

### SHAP explainers and instance-level attribution

To interpret the decisions of each trained classifier, a SHAP explainer is fitted on the training data of each fold using the `shap` Python library. SHAP values are then computed for a representative subset of test instances drawn from the held-out fold. This sampling step limits computational overhead while preserving the diversity of observed behaviours across benign and malicious traffic.

The output of this stage is a set of per-feature attribution values that quantify how each attribute contributes to the model output for a given instance. The same procedure is applied iteratively across all considered attack types (DoS, DDoS, BFA, U2R) and across the full set of supervised baselines adopted in the study.

### Global feature ranking and SHAP summary visualization

For each {attack, model} combination, the local SHAP attributions are gathered into a global importance ranking by computing the mean absolute SHAP value for each feature over the considered test subset. This aggregation yields an ordered list of attributes that most strongly drive model decisions in that specific binary setting.

SHAP summary (beeswarm)[12] plots provide a visual representation of both the ranking and the direction of the effect of each feature on the decision-making process. In these plots, each point corresponds to a test instance and is positioned according to its SHAP value for the given feature; the point color encodes the underlying feature value, making it possible to observe whether low or high values of an attribute tend to push the decision towards the benign or the attack class. Figure 5 provides an insight about the most important feature for the decision making process of SVM model on instances of the test fold. For completeness, the Appendix A provides the complete set of beeswarm plots for the DDoS experiment using all of the models investigated.

[12]Documentation available at: `https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/plots/beeswarm.html`.

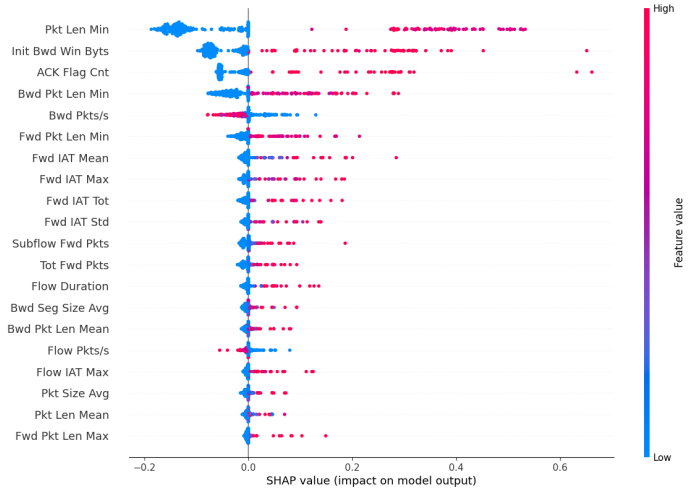



**Figure 5:** SHAP beeswarm plot for SVM model on InSDN dataset.

### SHAP Interpretation and Evasion Considerations for DDoS Traffic

The SHAP analysis demonstrates that, for all six classifiers examined in this research, DDoS discrimination can be attributed to a relatively small number of flow-level descriptors that capture *rate*, *directionality* and *size regularity*. The Shapley summary plots show that the dominant contributors were mainly packet-rate and packet-length descriptors, including the number of packets transmitted per second in the backward direction (`Bwd Pkts/s`), the overall packet rate within the bidirectional flow (`Flow Pkts/s`), the minimum packet length observed across the flow (`Pkt Len Min`) and the minimum packet length in the backward direction (`Bwd Pkt Len Min`). These features, together with related average-based and correlated descriptors, were the dominant contributors to the decision-making process of the classifier models, generally pushing the model output toward the malicious class when the traffic exhibited high-frequency exchanges and small, repeated payload sizes.

TCP-specific indicators, such as the number of packets with the ACK flag count (`ACK Flag Cnt`), and features that indirectly describe the

state of the transport connection, such as the initial window size in the backward direction (`Init Bwd Win Byts`), further contributed to separating flooding behaviours that induce atypical acknowledgment patterns or abnormal connection dynamics. Tree-based models showed more heterogeneous attribution distributions, consistent with non-linear feature interactions, whereas SVM and MLP presented sharper separations between benign and malicious contributions, suggesting a more stable reliance on these high-level rate-related features.

However, the same convergence of explanations also highlights a structural limitation: detectors that rely predominantly on rate- and size-based flow features are, in principle, susceptible to *imitation* and *traffic-shaping* strategies. An adaptive adversary could attempt to reduce detectability by keeping per-flow rates closer to benign ranges, by spreading load across many sources and flows so that `Flow Pkts/s` and `Bwd Pkts/s` do not have peaks, increasing variability in packet sizes and timing or shifting to attack modalities that do not strongly perturb the TCP indicators emphasized by the models, thereby weakening the diagnostic value of `ACK Flag Cnt` and window-related features. More generally, any evasion tactic that makes malicious traffic *statistically resemble* the background workload along these few dominant dimensions can degrade a classifier whose decision boundary is largely anchored to them. This motivated a deeper analysis on the most correlated features detailed in Subsection 3.2.4.

**Accuracy-constrained feature subset selection and retraining**

To evaluate whether the detection task can be solved with a more compact representation, the ranked list is used to construct progressively smaller feature subsets. Starting from the most important attribute, features are incrementally added following the SHAP ranking until the smallest subset is found that achieves at least a target accuracy threshold, set to $0.90$ in the analysis, under the same cross-validation protocol.

Once the subset size satisfying the target criterion is identified, the classifier is retrained using only the selected attributes, again under stratified 5-fold cross validation and with scaling fitted on training folds only.

This step produces a *reduced-feature* model that is directly comparable to the full-feature baseline, while offering lower dimensionality and potentially lower computational cost. Results in this subsection are summarized through tables reporting the selected subset size, the chosen features and the corresponding performance metrics.

Table 8 maps the feature with the label identified by CICFlowMeter in references that will be used in Table 9 for a better readability.

**Table 8:** Mapping between selected flow features and their reference identifiers. Adapted from Di Gennaro et al., 2024. © 2024 IEEE.

| **Feature** | **Ref.** | **Feature** | **Ref.** |
|---|---|---|---|
| Pkt Len Min | (1) | Bwd Seg Size Avg | (13) |
| Init Bwd Win Byts | (2) | Fwd Pkt Len Min | (14) |
| Bwd Header Len | (3) | Bwd Pkt Len Mean | (15) |
| Subflow Fwd Pkts | (4) | Bwd Pkt Len Std | (16) |
| Down/Up Ratio | (5) | Flow Duration | (17) |
| Bwd Pkts/s | (6) | CWE Flag Count | (18) |
| SYN Flag Cnt | (7) | Fwd Pkt/b Avg | (19) |
| Pkt Len Max | (8) | Fwd Byts/b Avg | (20) |
| Fwd Pkt Len Max | (9) | Bwd Seg Size Avg | (21) |
| Idle Mean | (10) | Bwd Pkt Len Min | (22) |
| Flow IAT Mean | (11) | Pkt Len Min | (23) |
| Pkt Size Avg | (12) | Fwd Pkt Len Std | (24) |

Table 9 shows the results obtained from the ML analysis with the reduced feature set and the references of the necessary features to obtain the target value of accuracy.

**Full vs. reduced feature-set comparison and implications**

Ultimately, the performance attained using the subset of features is compared to the reference performance attained with the entire feature set. The comparison is made on aggregate measures (accuracy, precision, recall, and $F_1$-Score) as well as on the specific entries of the confusion matrices, that allow to evaluate both the number of false positive classifi-

**Table 9:** Per-attack performance and selected feature reference identifiers by classifier. Adapted from Di Gennaro et al., 2024. © 2024 IEEE.

| Attack | Metric | SVM | DT | NB | KNN | RF | MLP |
|---|---|---|---|---|---|---|---|
| DDoS | Features | (1), (2) | (3) | (4) | (5) | (3) | (6) |
| | Pr | 0.97274 | 0.99602 | 0.97422 | 0.92105 | 0.99602 | 0.97039 |
| | Recall | 0.96894 | 0.99587 | 0.97074 | 0.89902 | 0.99587 | 0.97097 |
| | F1 | 0.96996 | 0.99594 | 0.97170 | 0.90100 | 0.99594 | 0.97054 |
| | Acc | 0.97010 | 0.99595 | 0.97182 | 0.90279 | 0.99595 | 0.97055 |
| DoS | Features | (1), (2), (7) | (8) | (2), (8) | (7) | (8) | (2), (7) |
| | Pr | 0.97390 | 0.98991 | 0.71429 | 0.49224 | 0.98991 | 0.97390 |
| | Recall | 0.70587 | 0.90035 | 0.98949 | 0.50000 | 0.90035 | 0.70587 |
| | F1 | 0.78565 | 0.94049 | 0.79469 | 0.49609 | 0.94049 | 0.78565 |
| | Acc | 0.99059 | 0.99653 | 0.97930 | 0.98448 | 0.99653 | 0.99059 |
| BFA | Features | (2), (7) | (8) | (2), (8) | (2), (7) | (8) | (2), (7) |
| | Pr | 0.88915 | 0.97578 | 0.59091 | 0.94928 | 0.97578 | 0.88915 |
| | Recall | 0.75774 | 0.66932 | 0.98980 | 0.99974 | 0.66932 | 0.75777 |
| | F1 | 0.80997 | 0.74923 | 0.64870 | 0.97316 | 0.74923 | 0.80997 |
| | Acc | 0.99716 | 0.99694 | 0.97970 | 0.99949 | 0.99694 | 0.99716 |
| U2R | Features | (2), (7), (9), (10), (11) | (12) | (2), (7), (13), (14), (15) | (2), (5), (7), (14), (16) | (17), (18), (19), (20), (21) | (2), (5), (7), (9), (10), (12), (15), (16), (21), (22), (23), (24) |
| | Pr | 0.99989 | 0.74993 | 0.87496 | 0.87496 | 0.99989 | 0.99993 |
| | Recall | 0.62500 | 0.74993 | 0.87496 | 0.87496 | 0.62500 | 0.75000 |
| | F1 | 0.69995 | 0.74993 | 0.87496 | 0.87496 | 0.69995 | 0.83330 |
| | Acc | 0.99978 | 0.99971 | 0.99985 | 0.99985 | 0.99978 | 0.99985 |

cations and false negative classifications for each class, to evaluate if the performance has been maintained at the same level (i.e., no change), decreased or even increased thanks to the elimination of noise and redundancy from the feature set.

Beyond raw performance, the comparative study will help identify methodological implications of the feature reduction process. Specifically, if a very small number of features are sufficient to achieve a level of high accuracy, it may indicate that the data contains many highly discriminative features or that there is little within-class variation. Both of these conditions can significantly decrease the difficulty associated with the learning problem, while at the same time decreasing the models' ability to adapt to shifts in the underlying data distributions. On the other hand, if there is a significant decrease in performance as the dimensionality of the data is decreased, it could indicate that the task requires a greater amount of descriptive information about the behaviour of the system being modeled; further, this may be indicative of a more realistic variability in the system's behaviour. These points of view are further supported by the SHAP visualizations and the specific distribution plots for the most influential attributes, both of which clearly illustrate the relationship between statistical separability, reliance on specific attributes of the input data and the observed classification outcomes.

### 3.2.5 Summary of empirical findings on InSDN

Table 9 lists for each attack class and classifier the compact subset of ***dominant*** features required to obtain an accuracy of at least $0.90$ when using the features, along with their respective performance metrics. Overall, the results show that with a sufficient amount of data available for training and testing, it is possible to accurately detect attacks (both by type of attack and by learning algorithm) using ***one to three features***, regardless of the model used.

A closer inspection of the results reveals that just one feature was sufficient to achieve high performance in detecting the DDoS class across all classifiers, except for SVM that needed two features to get similar per-

formance to what is achieved with the entire set of features. In addition, while the predominant feature differed from classifier to classifier, there were some similarities; specifically, the ***Bwd Header Len*** feature was chosen by both Decision Trees and Random Forests.

For DoS, BFA and U2R, several predominant features recur across models. Table 10 summarizes the occurrence counts of the three most recurrent features. Among the prevalent attributes, three stand out as particularly influential across the analyzed attacks and classifiers: *Init Bwd Win Byts*, the total bytes advertised in the initial backward TCP window, *SYN Flag Cnt*, the number of packets carrying the SYN flag and *Pkt Len Max*, the maximum observed packet length in the flow.

**Table 10:** Occurrence counts of the most recurrent predominant features across models (features selected more than twice within the reduced subsets). Adapted from Di Gennaro et al., 2024. © 2024 IEEE.

| Attack | Init Bwd Win Byts | SYN Flag Cnt | Pkt Len Max |
|---|---|---|---|
| Denial of Service (DoS) | 3 | 3 | 3 |
| Brute Force Attack (BFA) | 4 | 3 | 3 |
| User-to-Root (U2R) | 4 | 4 | 0 |

The prominence of *SYN Flag Cnt* for DoS, DDoS, brute-force and U2R can be interpreted in terms of the traffic patterns that characterize these threats. Although the operational goals differ, these scenarios share the tendency to generate repeated connection attempts or probing activity. In volumetric DoS/DDoS, the target is flooded to exhaust resources, regardless of whether the traffic sources are centralized or distributed. In brute-force attacks, repeated authentication attempts typically induce abnormal connection establishment dynamics. Similarly, U2R scenarios often start with an exploratory phase (e.g., reconnaissance and service probing), which can polarize the distribution of SYN-related events with respect to benign traffic. As a result, models can exploit this feature as a strong discriminator between normal and malicious instances.

At the same time, relying on *SYN Flag Cnt* alone would be brittle in adversarial settings. An attacker can reduce the detectability of SYN-

dominated patterns by combining heterogeneous flooding strategies (e.g., SYN, ICMP, UDP) or by adopting low-and-slow tactics that spread resource exhaustion over longer intervals. These considerations suggest that *SYN Flag Cnt* should be interpreted as a useful indicator, but not as a sufficient standalone detector.

The feature *Init Bwd Win Byts* can also exhibit anomalous behaviour in these scenarios, because resource pressure at the target may force the advertised window size to shrink, especially under sustained DoS/DDoS. Similar effects may arise for intense brute-force and reconnaissance activity, where repeated connections increase processing load. However, this signal can be attenuated through the same evasion strategies discussed above, reinforcing the need for multi-feature decision rules.

Finally, *Pkt Len Max* is frequently relevant for DoS and BFA. For DoS, this can reflect the packet-size configuration chosen by the attacker, although in principle different payload sizes can be used, reducing the universality of the cue. For BFA, instead, *Pkt Len Max* would not be expected to be inherently anomalous. To investigate this inconsistency, the variability of the feature was inspected, revealing that most BFA instances span over a very limited range of values for *Pkt Len Max*, whereas benign traffic spans a much broader range. Figure 6 shows an example of the the observed degeneracy that could explain why a small feature subset can separate normal traffic from BFA with high accuracy. Similar considerations may apply to other features and attack classes; however, for brevity, only this representative example is presented here.

### 3.2.6 Limitations of InSDN as a benchmark dataset

The findings highlighted in the previous section reveal numerous methodological weaknesses in InSDN's use as the primary reference dataset to evaluate SDN-based intrusion detection systems.

First, the fact that only one or two characteristics were sufficient to detect all attacks within a particular family is indicative of a significant bias within the distribution of the data. Features like *SYN Flag Cnt* and *Init Bwd Win Byts* show distinct separations between normal and abnor-

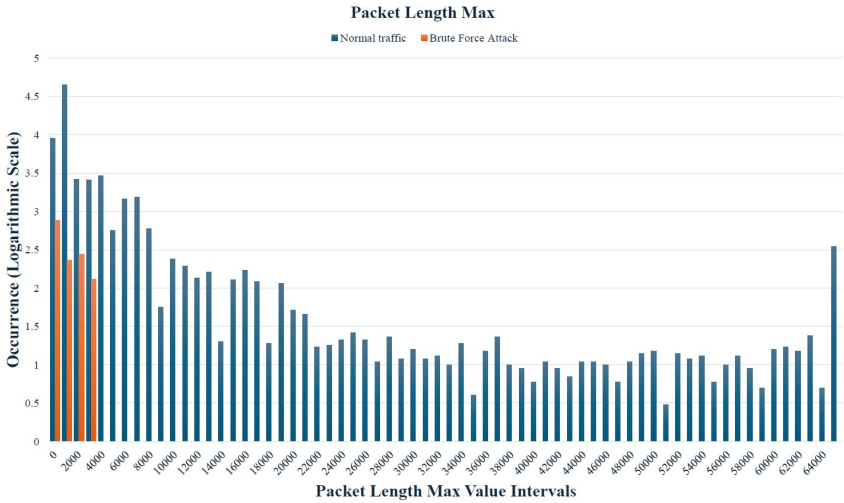



**Figure 6:** Example of feature variability between Normal traffic and Brute Force Attack traffic for Packet Length Max. Reprinted from Di Gennaro et al., 2024. © 2024 IEEE.

mal flows, to the extent that simple thresholding would allow for high performance. Although this simplifies the supervised learning problem, it also implies that detection models that have been trained using InSDN will likely focus on narrow signatures that can easily be circumvented by an intelligent adversary (i.e., by mixing various types of floods, by slowing down attacks or by creating packets that carefully mimic benign flag patterns).

Second, the preliminary study has shown that there exists limited variation in the features related to various attacks when compared to normal traffic. For instance, the consistent behaviour of *Pkt Len Max* for BFA flows suggests that the attack generation scripts employed during testing produced very regular behaviours, which are likely to fail to account for the variety of behaviours exhibited by actual real-world brute-force campaigns over time. Moreover, the traffic traces represent short-term experiments conducted in a controlled environment, rather than long-lasting, multi-day operations with diverse backgrounds of traffic.

Third, the basic SDN structure employed is quite simple. The testbed includes a single OVS switch VM, few virtual hosts and no additional network devices. This configuration does not accurately depict the complexities present in typical enterprise networks which include multiple switches, a variety of link bandwidths, segmented subnets and a combination of clients, servers and IoT-like devices. Consequently, the dataset does not expose some of the key issues present in large-scale networks, including congestion on individual links and routing changes.

Fourth, the InSDN dataset only contains flow-level statistics derived from packet captures and, therefore, are limited to the perspective provided by the data plane. No explicit control plane metrics (e.g., OpenFlow statistics, controller events, flow modification rates), nor host-level resource indicators (e.g., CPU, memory, process level metrics) are available for inclusion. Thus, it is impossible to develop and evaluate multi-plane detection methods that combine information from the controller, switches and end-hosts.

Finally, the number and diversity of attack types are limited. Although DoS, DDoS, BFA and U2R are valid, there exist numerous other relevant threats in SDN that include flow-rule manipulation by malicious applications, stealthy low rate attacks and lateral movement across network segments that are not included. Therefore, the ability to study the generalization of IDS models across a wide range of threats is significantly limited, and the robustness of IDS models against a wide array of threats cannot be tested beyond a few standard scenarios.

### 3.2.7 From InSDN gaps to KRONOS-SDN design requirements

Despite these limitations, the InSDN analysis plays a crucial role in the methodological pipeline on the thesis. First, it allows to validate the basic machine learning and feature-importance tools on a well-defined and widely referenced dataset, ensuring that implementation choices (preprocessing, model selection, cross-validation, SHAP configuration) reproduce correctly state-of-the-art results. Second, by exposing the struc-

tural weaknesses of InSDN, it provides concrete guidance for the design of a new SDN dataset that addresses those gaps.

In particular, the following design requirements emerge from the discussion above and should directly motivate the choices about the design of datasets for ML-based IDS in SDN architectures:

- **[DR I] Increased intra-class variability**. This criterion is satisfied if each major attack class is implemented through more than one execution profile, for example by varying tools, rates, payload sizes, durations or timing patterns, and if the attack traffic is interleaved with realistic background traffic.

- **[DR II] Broader attack coverage**. This criterion is satisfied if the dataset includes attack families beyond volumetric DoS/DDoS and brute-force attempts, covering also stealthy low-and-slow behaviours and multi-stage campaigns combining reconnaissance and exploitation.

- **[DR III] Richer and more complex topology**. This criterion is satisfied if the experimental environment includes multiple network segments, switches and heterogeneous host types, such as clients, servers, management nodes, rather than a minimal flat topology.

- **[DR IV] Extended feature space across planes**. This criterion is satisfied if the dataset includes, or enables the extraction of, information beyond packet-derived flow statistics, such as control-plane indicators, OpenFlow-related counters or host-level utilization metrics including CPU, memory, I/O and entropy.

- **[DR V] Longitudinal collection**. This criterion is satisfied if traffic is collected over multiple days and includes different workload phases, such as daytime activity, nighttime activity, workload peaks and maintenance-like periods.

Each criterion is considered satisfied when the resulting dataset exhibits the corresponding observable property, such as the presence of

multiple attack variants, multiple attack families, a segmented topology, cross-plane telemetry sources or multi-day traffic captures.

In summary, the baseline study on InSDN reveals that, while traditional flow-based models can achieve excellent performance on this dataset, such performance is largely driven by a small number of highly discriminative but potentially brittle features set. This observation, together with the constraints inherent in the original testbed and traffic generation process, motivates the construction of a new, large-scale, cross-plane dataset designed to stress more realistic and challenging detection conditions.

# Chapter 4

# Design and analysis of the KRONOS-SDN dataset

Building on the limitations identified in the analysis of the InSDN dataset presented in Section 3.2, this part of the thesis details the methodology adopted for the design, generation and analysis of the proposed Kill-chain Realistic Operational Network for Optimized Security in SDN (*KRONOS-SDN*) dataset. The goal is threefold:

- to construct a realistic, temporally rich SDN testbed that jointly fills the gaps of other benchmark dataset, measured through the DRs presented in Subsection 3.2.7;
- to provide a simulated environment that could be used as cyber-range for further research in this field;
- to provide a transparent, reproducible analysis pipeline covering statistical characterization, supervised learning baselines and explainable AI (XAI) techniques.

The remainder of the chapter is structured as follows. Section 4.1 introduces the design objectives and motivates the key architectural choices. Section 4.2 then describes the KRONOS-SDN testbed, covering both its physical infrastructure and logical SDN topology. Section 4.3 details

the benign workload model together with the orchestrated attack campaigns, while Section 4.4 explains the data acquisition workflow, including feature extraction and ground-truth labelling. Section 4.5 presents the pre-processing pipeline and the adopted feature analysis and selection strategy. The supervised baseline experiments and their outcomes are reported in Section 4.6. Finally, Section 4.7 discusses the SHAP-based interpretability analysis and derives methodological implications for SDN-oriented IDS design.

## 4.1 Design objectives and rationale

The KRONOS-SDN design was driven by the comparison among SDN-based intrusion detection datasets and specifically from the baseline study on InSDN described in Section 3.2 combined with related benchmarks. These datasets have stimulated ML-based IDS research in SDNs, but, as described in Chapter 3, have also shown structural constraints that prevent them from being used for the specific objectives of this thesis and provided a precise set of design requirements (DRs) for a new dataset.

First, attack traces are frequently staged as short, isolated bursts and implemented with limited variability in implementation, are typical for most of these datasets. This characteristic can lead to degenerate feature distribution and highly separable class boundaries such that an attacker could be detected unrealistically easily using simple decision rules based on one feature alone. This directly motivates **[DR I]** which aims at increasing intra-class variability through multiple attack variants and mixing of these ones with realistic background traffic. Similarly, many of the benchmark datasets limit themselves to a relatively small catalog of attacks including primarily volumetric DoS/DDoS and include very few or no low-and-slow stealthy attacks, nor do they provide much if any representation of multi-stage campaigns. Therefore, this lack of attack coverage provides motivation for **[DR II]**, i.e., to expand attack coverage to include reconnaissance, exploitation, and post-exploitation behaviours.

Second, a large proportion of current SDN benchmarking studies rely on reduced testbeds that are often built on minimal topologies using a single controller, a single switch and a very few homogeneous nodes. These models don't model real world enterprise segmented environments and also have limited variety in benign interaction types and failure types, so this motivates **[DR III]** consisting of a richer and more complex topology with heterogeneous node types and multiple network subnets.

Third, most datasets expose traffic-derived, flow-level features only, without providing control-plane measurements (or suitable proxies such as OpenFlow counters) and host utilization statistics that are routinely available in operational environments through standard monitoring tooling. This prevents the evaluation of cross-plane and multi-modal detection strategies and directly motivates **[DR IV]**, i.e., an extended feature space across planes that combines packet/flow statistics with host-level signals such as CPU, memory, I/O, entropy and interface counters.

Fourth, there exists a significant shortage of "longitudinal continuity" within current SDN datasets; traces are usually captured for short durations, or as separate disconnected events. Therefore, understanding long-term temporal patterns, daily workload variations, and slow trend changes are very challenging using these types of datasets. The motivation for this is **[DR V]**, capturing network traffic for a duration of at least a week while being exposed to various operational regimes.

Against this background, KRONOS-SDN is conceived as an operationally oriented benchmark that satisfies the above requirements through a coherent set of design choices. It operationalizes **[DR I]** and **[DR II]** by implementing attack scenarios in multiple variants and by orchestrating multi-stage campaigns interleaved with realistic background services. It addresses **[DR III]** by combining an SDN controller with multiple OVS instances and virtual machines representing business clients, servers and supporting services, enabling attacks to be observed from different vantage points. It enables **[DR IV]** by coupling flow-level statistics extracted from packet captures with host-level utilization metrics for selected nodes, supporting multi-modal IDS designs that exploit both traffic and resource-usage evidence.

Ultimately, it provides **[DR V]**, with a weekly collection window allowing for the capture of both diurnal patterns as well as workload variability and maintenance-like periods. As all stages from traffic generation and packet capture to feature extraction, labelling and downstream ML analysis have been fully automated and controlled through versioning to provide repeatable results and an open platform to support systematic extensions.

To summarize, the InSDN-driven baseline analysis does not simply serve to inspire KRONOS-SDN on an abstract level; rather, it provides concrete design DRs that directly influence the dataset's time scale, topology, measurement planes and attack/workload orchestration. Section 4.2 describes how these specifications will be realized within the KRONOS-SDN testbed and data generation process.

## 4.2 SDN testbed architecture

### 4.2.1 Physical infrastructure and virtualization

The KRONOS-SDN testbed is deployed on a dedicated server equipped with multi-core x64 CPUs, 80 GB of RAM and SSD storage. Virtualization is provided through a type-2 hypervisor, with each functional role of the SDN network mapped to multiple virtual machines (VMs). Each VM is configured with a minimal yet realistic software stack (Linux distribution, Open vSwitch, ONOS, web and SSH servers, DNS and other relevant services) and is attached to virtual networks that mirror an enterprise-like segmentation into user, server and management subnets. The architecture is configured to provide accurate wall-clock time to each VM through Network Time Protocol (NTP), enabled to keep the internal clocks aligned with a NTP virtual server, thus ensuring consistent timestamps across packet captures and host-level logs.

### 4.2.2 Logical SDN topology

The network topology is described in Figure 7. The focal point is the clear separation between the control and data planes interconnected via

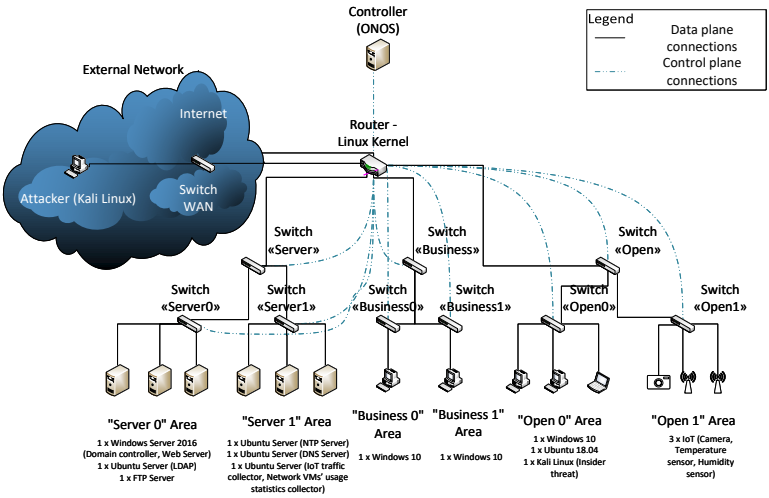



**Figure 7:** Network topology of KRONOS-SDN architecture.

the OpenFlow protocol (Open Networking Foundation, 2014a), as this architecture is a reference defined by the Open Networking Foundation. The ONOS controller maintains a global view of the network and installs forwarding rules in the OVS instances via OpenFlow channels. The data plane is composed of the Open vSwitch instances attached to the Open, Business and Server areas. The traffic is collected at each active networking element, excluding the Linux-based router.

The resulting logical network topology is organized into five functional areas.

- **Server Area**, which hosts the local server-side environment and provides both core infrastructure services and application-level functions;

- **Business Area**, composed of enterprise-like clients that primarily consume the services delivered by the Server Area;

- **Open Area**, populated by heterogeneous clients with Internet connectivity, where nodes also export sensor/telemetry data toward a dedicated collector server;

- a network segment reserved for the **ONOS Controller**, responsible for orchestrating and managing the SDN switching elements;

- a separate **External Network** segment, outside the SDN management scope, used to emulate untrusted external access.

This topology supports both east/west and north/south communication patterns: business clients access services on the servers located in the Server area, the controller manages flows over the Open switch and attack traffic can originate either from internal compromised hosts or from an external WAN segment.

The five logical areas, each designed to reproduce a distinct operational role and to generate traffic and telemetry patterns that are representative of realistic deployments. The mentioned segmentation is also instrumental to enforce trust boundaries and to support controlled experimentation, since attacks and benign workflows can be activated in specific zones while their effects are observed across the infrastructure.

The *Server Area* concentrates shared services that typically act as backbone dependencies in enterprise networks. It hosts centralized identity functions (e.g., a domain controller and directory-backed authentication), core infrastructure services such as DNS and NTP and service endpoints that sustain client-server workloads. The same area also includes an IoT collector, which aggregates periodic telemetry exports and enables cross-area data flows that are useful for correlation and context-aware analysis.

The *Business Area* emulates internal corporate endpoints. It comprises domain-joined clients distributed over two segments, capturing the effect of departmental separation while maintaining realistic intra-campus

communication. Clients execute routine workflows toward the internal services and generate recurring authentication and directory interactions, thereby stressing identity-dependent traffic patterns and producing benign baselines that are non-trivial and time-varying.

The *Open Area* represents reduced-trust zones, closer to the network edge. It hosts heterogeneous endpoints and IoT nodes that are intentionally exposed to more variable operating conditions. Clients in this area are granted Internet access, enabling outbound traffic patterns and external reachability assumptions that differ from the internal segments. IoT devices periodically export telemetry toward the IoT collector, creating regular cross-area communications that can be leveraged to study hybrid detection and cross-plane correlations.

The *Controller Segment* hosts the ONOS SDN control plane. Through the OpenFlow channel, ONOS centrally manages the internal OVS instances, maintains topology and host awareness and enforces policies via intents and flow-rule programming. This segment also serves as a natural integration point for observability (events, counters) and, when applicable, for closed-loop responses where detection outcomes can be translated into mitigation actions.

Finally, the *External Network (WAN)* acts as an untrusted, Internet-side environment. It is outside the controller's direct management and provides a boundary context to test trust separation and the exposure of internal services. In experimental campaigns, this segment is used to generate external-origin traffic, including scanning and abuse patterns, thus enabling realistic evaluation of IDS detection performance and mitigation policies at the network perimeter.

## 4.3 Traffic generation and attack campaigns

### 4.3.1 Benign workloads

Benign traffic was generated to emulate typical enterprise usage patterns, also considering workload peaks during the week. The generation process was not performed manually on a per-connection basis. In-

stead, benign activities were produced through a set of programmatically scheduled client-side scripts executed on the hosts belonging to the Business Area and the Open Area. These scripts periodically triggered common network operations, such as name resolution, time synchronization, web requests, authentication-related exchanges and file transfers, according to a predefined weekly schedule.

The scripts relied on standard client applications and command-line utilities available on the involved machines. For example, DNS and NTP traffic was produced through periodic resolution and synchronization requests, web traffic was generated through repeated HTTP/HTTPS accesses to internal or external services, and FTP traffic was generated through authenticated file-transfer sessions with the servers located in the Server Area. In the Business Area, the generated activities were mainly directed toward internal enterprise services, such as the DNS/NTP server, the web server, the FTP server and Active Directory-related services. In the Open Area, benign traffic was instead designed to reproduce less controlled user and device behaviour, including external web browsing, DNS/NTP requests, email-related communication, application traffic and IoT-like telemetry.

The generation schedule was defined over the whole seven-day acquisition period. Each benign workload was associated with specific time windows and repetition rates, so that traffic intensity varied according to a diurnal pattern: higher during working hours, lower during nighttime periods and reduced during weekends. To avoid unrealistically monotonic traffic rates, an additional randomization component was introduced in the activity scheduling process. In particular, the actual execution rate of each benign activity was not fixed deterministically, but was sampled within predefined ranges depending on the time of day. Daytime intervals were associated with higher activity-rate ranges, whereas nighttime intervals were associated with lower ranges. This mechanism preserved the intended daily workload profile while introducing natural variability in the timing and intensity of benign traffic.

Table 11 summarizes the main features of the traffic generated by the business clients and consequently by the servers that offer the services

from the *Server Area* side.

**Table 11:** Main categories of network traffic generated by the Business Area clients.

| Traffic Type | Key parameters / factors |
|---|---|
| DNS Queries | UDP/53; internal + external name resolution; client → DNS server (Server Area). |
| NTP Time Sync | UDP/123; periodic clock synchronization; endpoints → NTP server (Server Area). |
| HTTP Web Traffic | TCP/80; web/service requests (e.g., GET/POST); client ↔ server. |
| Active Directory Services | Authentication and directory traffic: LDAP (TCP/389) and DNS-based service discovery (UDP/53). |
| FTP File Transfers | FTP control TCP/21; data channel TCP/20 (active) or negotiated ephemeral ports (passive); authenticated sessions. |

Table 12 enumerates the traffic generated by the benign clients located in the *open area*. Overall, the benign workload generation process com-

**Table 12:** Main categories of network traffic generated by the Open Area clients.

| Traffic Type | Description |
|---|---|
| DNS Requests | Domain name resolution queries to external DNS servers, over UDP port 53. |
| NTP Synchronization | Periodic clock synchronization with external time servers, over UDP port 123. |
| HTTP/HTTPS Requests | Web browsing and service communication using HTTP (TCP 80) and HTTPS (TCP 443) while web surfing. |
| Email Traffic | Communication with external mail servers via SMTP. |
| Generic Application Traffic | Data exchange from applications, consisting of web services, IoT telemetry and so on, typically using dynamic TCP/UDP ports. |

bined service-specific traffic patterns, time-dependent execution schedules and activity rates that incorporate a modest degree of randomness alongside the usual patterns of working-hour rates. This design avoids overly regular traffic profiles and introduces realistic variability in both traffic volume and host resource utilization, which is critical to assess IDS robustness under workload fluctuations.

### 4.3.2 Attack scenarios

Malicious activities are orchestrated as multi-stage campaigns rather than isolated episodes. The attack catalogue includes the attacks detailed in Table 13.

**Table 13:** Attack classes implemented in the experimental campaign and their targets.

| Attack class | Implementation and target |
|---|---|
| Reconnaissance | Host discovery and port scanning with `nmap`, targeting both the *Server Area* and *ONOS* VMs. |
| Brute-force authentication | SSH password guessing using tools such as `hydra`. |
| Application-layer DoS | HTTP flooding and Slowloris-style attacks against the web server. |
| Network-layer / OpenFlow-specific DoS | ICMP flooding (Smurf-like) and rule-table exhaustion attacks targeting the controller via the *Open* switch. |
| Amplification attacks | Reflection/amplification attacks (e.g., LDAP amplification) against services exposed in the WAN-facing segment. |

Attack instances are distributed across the seven days with varying duration and intensity. Some attacks are short bursts embedded in normal traffic, while others last for extended intervals (tens of minutes) to simulate persistent DoS activity. For each attack, the start and end times, the attacker IP address, the victims and the attack label are recorded in a dedicated ground-truth file. This schedule is later used for labelling both flow records and host-level statistics.

The malicious traffic is introduced through two attack nodes, both based on Kali Linux. One node is located outside the network representing the area under ONOS control, that is to say within the WAN segment, while the other is located within the 'Open0' area, representing an internal threat. These attackers are used to perform various types of cy-

ber attacks, such as network reconnaissance, spoofing, denial-of-service (DoS), DNS and LDAP exploitation.

This configuration allows the emulation of complete attack chains and the collection of diverse and realistic network traffic conditions for use in IDS training and validation. The hybrid structure of the test bed, which combines legitimate business activities with different attack scenarios, supports the study of complex threat dynamics, anomaly detection patterns and adaptive security mechanisms. A compact summary of the tools and their role in the traffic generation is reported in Table 14.

**Table 14:** Summary of attack-generation tools used in the traffic plan.

| Tool | Role in the scenario |
|---|---|
| Nmap[9] | Network reconnaissance and service enumeration to emulate early-stage scanning and host discovery. |
| Hydra[10] | Automated credential guessing to simulate brute-force attacks against SSH and web administration interfaces. |
| Scapy[11] | Packet crafting and injection, including custom/malformed OpenFlow packets to test SDN control-plane robustness. |
| hping3[12] | Low-level packet generator used for SYN/UDP floods and other volumetric tests against the ONOS controller (OpenFlow port). |
| nping | Flooding and generic packet generation for background noise and lighter UDP-based attack streams. |
| Slowloris[13] | Application-layer "low-and-slow" DoS against web servers by exhausting connection resources. |

### Rationale for attack selection

The KRONOS-SDN attack catalogue was selected to mirror common enterprise intrusion workflows and stress SDN-specific choke points, thus providing realistic and diversification for both flow-based and cross- plane IDS evaluation [DR I]. First, *reconnaissance*, with host discovery and port scanning, is included because it is a recurrent early-stage activity used to

[9] https://nmap.org/book/man.html
[10] https://www.kali.org/tools/hydra/
[11] https://scapy.net/
[12] https://www.kali.org/tools/hping3/
[13] https://www.f5.com/glossary/slowloris

enumerate reachable assets and exposed services before exploitation and it is widely recognized as a practical precursor to multi-stage compromise (Pittman, 2023). Second, *brute-force authentication*, e.g., SSH password guessing, is modeled because credential attacks remain a prominent access vector in real-world incidents (European Union Agency for Cybersecurity (ENISA), 2024) and because SSH services are persistently targeted at Internet scale, often using off-the-shelf tools and large distributed infrastructures (Verizon DBIR Team, 2024; S. K. Singh et al., 2024).

Third, the dataset combines *application-layer DoS*, that is HTTP flooding and Slowloris-style low-and-slow patterns, with network-layer DoS and amplification mechanisms. This choice reflects the fact that modern availability attacks are frequently multi-vector, mixing volumetric pressure with protocol- and application-level resource exhaustion to evade simplistic thresholds and maximize disruption (Yoachimik et al., 2024). Amplification scenarios are explicitly represented because reflection remains operationally relevant and has been repeatedly observed in the wild as a cost-effective way to scale attack traffic, thereby complementing direct flooding streams (Lumen Black Lotus Labs, 2022; Health Sector Cybersecurity Coordination Center (HC3), 2024).

Finally, DoS behaviour specifically related to SDN is included to highlight the risks due to interactions between the controller and switch and for representing resource constraints for forwarding activity. Control-channel saturation and table exhaustion have been identified as two of the most significant DoS attacks on SDNs when faced with a malicious volume of traffic. The inclusion of these DoS behaviours have driven additional research into detection and defense strategies for SDNs in recent years (Su et al., 2024; Tang et al., 2024b; Shen et al., 2023; Tang et al., 2024a). The ability to include both types of attackers (external and internal) as well as their use of multi-phase campaign (from discovery to denial-of-service), allows the data to display greater variability within each class and a more accurate representation of time-evolution than isolated attack bursts, while also allowing for detectable effects at different points of observation.

## 4.4 Data acquisition, feature extraction and labelling

### 4.4.1 Packet capture and flow export

For each monitored VM (ONOS, Open, Business, Server), packet captures are collected at the network interface level using `tcpdump`. To facilitate processing and parallelisation, the full-day traces are segmented into four-hour time windows with a naming convention of the form:

`dump_{Machine}_{Day}_{StartHour-EndHour}.pcap`.

Flow-level features are extracted from the PCAP files using CICFlowMeter (Sharafaldin et al., 2018b), which has become a de facto standard for flow-based IDS datasets. For each flow, CICFlowMeter yields more than 80 statistical features including flow duration, packet counts and rates, byte counts, inter-arrival times, average and maximum packet lengths, header flags and measures of burstiness and idle time.

In KRONOS-SDN, flow extraction is performed per VM, generating CSV files that follow a consistent naming scheme and can be mapped back to the original PCAPs. This representation is used both for descriptive traffic analysis and for training machine learning models.

### 4.4.2 Host-level utilisation metrics

To complement flow statistics with system-level context, the `collectd` monitoring daemon is deployed on selected VMs, primarily the ONOS controller and the Server. The agent periodically samples CPU usage (user, system, idle, iowait), memory utilisation, disk I/O counters, network interface statistics (bytes and packets sent/received) and system load averages. Entropy-based indicators of resource usage are also computed to capture variability in time.

Metrics are exported in CSV format with timestamps aligned to the same time base as the packet captures. For subsequent analysis, per-interval aggregates are derived (e.g., mean CPU utilisation every 10 s

windows) and usage statistics from multiple cores or disks are aggregated into single indicators when appropriate.

### 4.4.3 Time alignment and day indexing

A crucial step in the dataset preparation is the alignment of timestamps across flows, PCAPs and host metrics. All machines are synchronised via NTP; nevertheless, validation scripts verify the consistency of clock offsets at dataset generation time. Each record (flow or utilization sample) is assigned:

- a **UNIX epoch timestamp** in UTC;
- a **local time** representation (Europe/Rome timezone) for interpretability;
- a **day index** in the range 1–7, mapping chronological dates to experimental days.

This indexing enables per-day analysis and supports cross-referencing with the ground-truth attack schedule.

### 4.4.4 Ground-truth labelling

The attack schedule is stored in a CSV file containing the attacker IP address, the start and end timestamps of each malicious episode, the affected machines and the attack type. Flow-level labels are generated by joining the flow records with this schedule based on timestamp overlap and IP matching. Each flow is thus assigned one of the following labels:

- `Benign`;
- a specific attack type (e.g., `SSH-Bruteforce`, `HTTP-DoS`, `OpenFlow-DoS`, `LDAP-Amplification`).

For the purposes of this chapter, labels are further aggregated into a binary variable (`Malicious` vs. `Benign`) to simplify the initial evaluation of baseline classifiers. Subsequent chapters will revisit multi-class and hierarchical labelling schemes.

## 4.5 Pre-processing and statistical feature analysis

The raw flow CSV files contain a mixture of categorical and numerical attributes, some of which can be grouped according to their semantic meaning. Before training machine learning models, the following pre-processing steps are applied:

1. **Cleaning and filtering.** Records with missing values, inconsistent labels or extreme outliers caused by capture artefacts are discarded. Zero-variance features are automatically removed per machine.

2. **Encoding.** Categorical features such as protocol type and TCP flags are encoded into numerical form using one-hot or ordinal encoding, depending on their cardinality and semantics.

3. **Scaling.** Numerical features are standardised to zero mean and unit variance on the training set to stabilise optimisation for gradient-based models. The same transformation is then applied to the test data.

4. **Train–test splitting.** For each monitored machine, the flows are split into training and test sets, preserving temporal ordering where required, for example in robustness experiments, or using stratified sampling to respect class proportions.

After pre-processing, a preliminary statistical screening was performed to estimate the relevance of individual flow features with respect to the traffic labels and to identify redundancy patterns before the application of learning-based detectors. Two complementary criteria were adopted: *Mutual Information* (MI)[1] and the *ANOVA F-test* (Kissell et al., 2017). MI measures statistical dependence between a feature and the label without assuming linearity, thus capturing generic, possibly non-linear, associations. The ANOVA F-test evaluates whether the mean value of a feature differs significantly across the groups induced by the class labels,

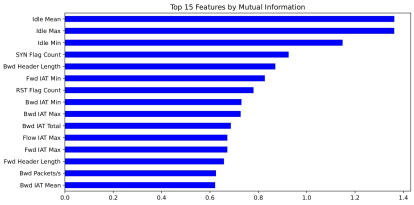
[1]Overview provided at: `https://www.sciencedirect.com/topics/computer-science/mutual-information`.

i.e., whether between-class variability dominates within-class variability (Ebrahem et al., 2025; Sarhan, 2024; Rana et al., 2022). Although based on different assumptions, the joint use of MI and ANOVA provides a robust, model-agnostic indication of potentially discriminative attributes.

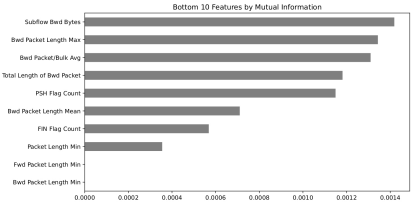
Figures 8a and 8b report, for the ONOS virtual machine, the features ranked by MI (top 15 and bottom 10, respectively). The highest MI scores are consistently attained by idle-time descriptors (e.g., *Idle Mean*, *Idle Max*, *Idle Min*), which summarize the silence intervals between consecutive packet bursts within the same bidirectional flow. Intuitively, such statistics capture temporal regularities of communication: benign applications often exhibit relatively structured inter-burst gaps (e.g., periodic keep-alives or user-driven interactions), whereas attack traffic may alter these patterns either by compressing idle gaps (high-throughput scanning or brute-force attempts) or by introducing irregular spacing (rate-controlled and stealthier behaviours). Anyway, just because mutual information (MI) value is high, it doesn't mean that one idle descriptor would be sufficient to have a trusted benign–malicious separation; there could be some overlap of the distributions between the macro classes.

In contrast, the least-informative MI-ranked features are almost entirely related to bulk-transfer descriptors; therefore these features appear to have little or no discriminative power within the given workload. The findings above are consistent with the fact that, under normal network traffic, bulk-transfer sessions are not dominant and under which the malicious traffic set does not consistently depend on large volume/long lived flows that can differentiate bulk statistics from other flows.

A consistent picture emerges from the ANOVA ranking. Figures 9a and 9b show the top 15 and bottom 10 features according to the ANOVA F-test for ONOS, with substantial agreement with the MI-based ordering. This cross-test consistency supports the interpretation that idle- and inter-packet temporal descriptors represent a statistically salient dimension in the dataset, while several bulk-related metrics remain largely uninformative. To assess feature inter-dependencies, the correlation heatmap in Figure 10 is used to visualize pairwise relationships across all attributes. Distinct clusters of strongly correlated features are observed,

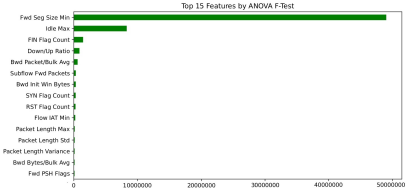



**(a)** Top 15 features ranked by MI.

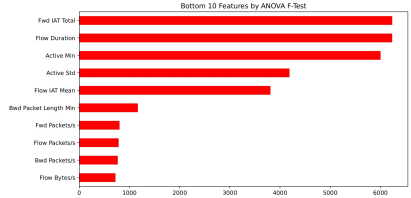



**(b)** Bottom 10 features ranked by MI.

**Figure 8:** Top 15 features (a) and bottom 10 features (b) ranked by Mutual Information for the ONOS controller datasets.




**(a)** Top 15 features ranked by ANOVA F-Test.




**(b)** Bottom 10 features ranked by ANOVA F-Test.

**Figure 9:** Top 15 features (a) and bottom 10 features (b) ranked by ANOVA F-test score for the ONOS controller dataset.

indicating redundancy among groups of descriptors derived from similar underlying counters (e.g., multiple variants of inter-arrival and idle statistics). These clusters motivate dimensionality reduction or informed feature pruning to mitigate redundancy and to improve computational efficiency during model training and inference. In order to further contextualize the statistical rankings, two representative features were inspected through per-class distributions (Figure 11). First, *Idle Max*, one of the most relevant attributes ranked for MI and resulting correlated in a direct way with the label as shown in Figure 10, demonstrates partially overlapping ranges between benign and malicious traffic, which limits its standalone value for binary detection. In the meanwhile, more structured differences can arise across individual attack categories, suggesting that idle extremes could assist in discriminating between the malicious

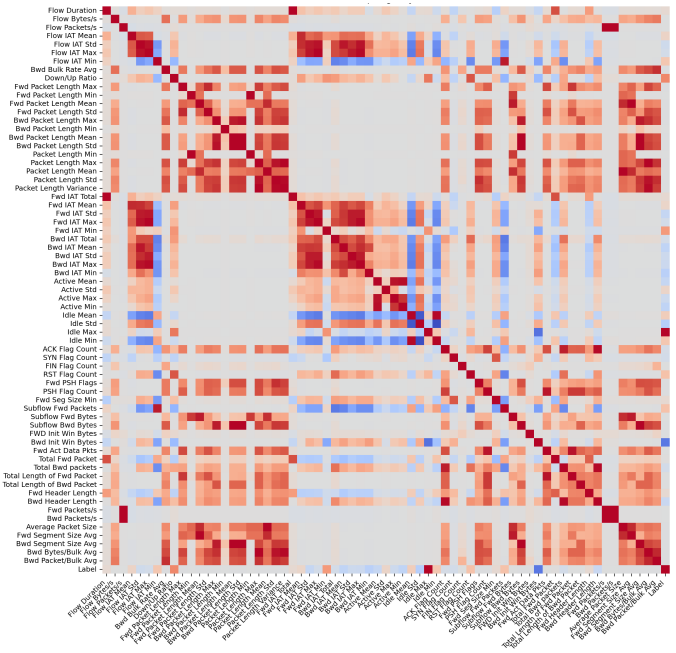
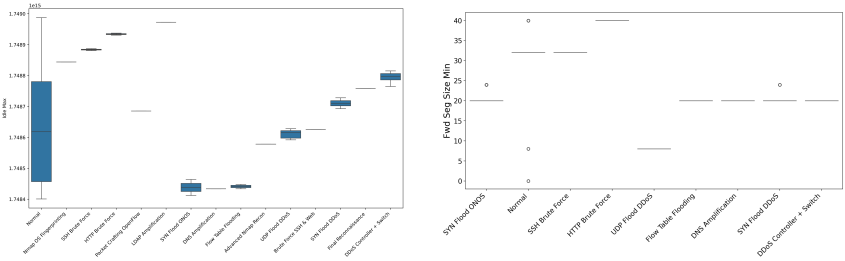



**Figure 10:** Graphic of the heatmap for the ONOS virtual machine dataset.

classes, but not necessarily between the benign and malicious classes. This interpretation is consistent with the downstream SHAP based attribution results presented in Section 4.7 where *Idle Max* does not have a significant influence on the decision making process for all classifiers and ranks high only for certain classifiers (as shown for SVM in Figure 12). Second, *Fwd Seg Size Min*, ranked as the first feature by ANOVA in this context, shows substantial overlap in several classes, which reinforces that high ANOVA relevance does not always equate to clear separation between classes without evaluating the multivariable relationships between these variables. Finally, a set of features with constant value across the ONOS traces is identified (e.g., *Bwd PSH Flags*, *Bwd URG Flags*, *Fwd Bytes/Bulk Avg*, *Fwd Packet/Bulk Avg*, *Fwd Bulk Rate Avg*, *Subflow Bwd Packets*, consistently equal to zero). Such attributes provide no discriminative




**(a)** Idle Max (Mutual Information) **(b)** Fwd Seg Size Min (ANOVA F-test)

**Figure 11:** Distribution for each label of two key flow features computed on the ONOS VM: (a) Idle Max and (b) Fwd Seg Size Min.

information and can be safely removed from subsequent analyses. Their constancy is compatible with expected operational patterns (rare usage of PSH/URG flags and lack of sustained bulk transfers in the considered workload), rather than indicating a bias in the acquisition process.

## 4.6 Machine learning analysis

### 4.6.1 Supervised benchmark models on the ONOS controller

A supervised learning benchmark was carried out on the ONOS virtual machine by framing intrusion detection as a *binary* classification problem, where all attack instances are aggregated into a single `Malicious` class and contrasted against `Normal` traffic. Two training regimes were considered to quantify the impact of class imbalance:

- **Balanced training set**, obtained by undersampling the majority class in the training folds;
- **Unbalanced training set**, preserving the original class distribution.

The benchmark covers widely adopted baselines, namely Naïve Bayes (NB), $k$-Nearest Neighbors (KNN), Decision Tree (DT), Random Forest

(RF), AdaBoost, Support Vector Machine (SVM) and a Multi-Layer Perceptron (MLP). The parameter configuration used for the experiments is summarized in Table 15. The models were implemented with `scikit-learn`[2].

**Table 15:** Benchmark models and parameter configuration (ONOS).

| Model | Key settings |
|---|---|
| **Naïve Bayes (NB)** | Smoothing variance $= 1 \times 10^{-9}$; priors: none. |
| **Multi-Layer Perceptron (MLP)** | Single hidden layer (100 neurons); activation: ReLU; solver: Adam; $\alpha = 0.0001$; learning rate init $= 0.001$; max iter $= 200$; shuffle: true; momentum $= 0.9$; early stopping: no. |
| **AdaBoost** | Base estimator: decision tree (depth $= 1$); estimators $= 50$; learning rate $= 1.0$; algorithm: SAMME.R. |
| **Decision Tree (DT)** | Criterion: Gini; splitter: best; max depth: none; min samples split $= 2$; min samples leaf $= 1$; class weights: none. |
| **Random Forest (RF)** | Trees $= 100$; criterion: Gini; bootstrap: on; class weights: none. |
| **K-Nearest Neighbors (KNN)** | Neighbors $= 5$; weights: uniform; metric: Euclidean. |
| **Support Vector Machine (SVM)** | $C = 1.0$; kernel: linear; probability: false; tolerance $= 10^{-3}$. |

**Table 16:** ONOS performance under balanced and unbalanced training ($F_1$, accuracy, precision and recall).

**ONOS (Balanced)**

| Model | F1 | Acc | Prec | Rec |
|---|---|---|---|---|
| NB | 0.9955 | 0.9955 | 0.9955 | 0.9955 |
| MLP | 0.9955 | 0.9955 | 0.9955 | 0.9955 |
| AdaBoost | 0.9977 | **0.9977** | **0.9978** | 0.9977 |
| DT | 0.9996 | 0.9955 | 0.9955 | 0.9955 |
| RF | **0.9998** | **0.9977** | 0.9977 | **0.9978** |
| KNN | 0.9865 | 0.9865 | 0.9868 | 0.9865 |
| SVM | 0.9865 | 0.9865 | 0.9868 | 0.9865 |

**ONOS (Unbalanced)**

| Model | F1 | Acc | Prec | Rec |
|---|---|---|---|---|
| NB | 0.7210 | 0.9994 | 0.6427 | 0.9907 |
| MLP | 0.9840 | **1.0000** | 0.9907 | 0.9775 |
| AdaBoost | **0.9977** | **1.0000** | 0.9977 | **0.9977** |
| DT | 0.9955 | 0.9977 | **1.0000** | **0.9977** |
| RF | **0.9977** | 0.9977 | **1.0000** | **0.9977** |
| KNN | 0.9875 | **1.0000** | 0.9909 | 0.9842 |
| SVM | 0.9400 | **1.0000** | 0.9944 | 0.8964 |

To control computational cost while preserving robustness, stratified $k$-fold cross-validation with $k = 5$ was adopted, maintaining the class ratio within each fold. Performance is reported in terms of accuracy, precision, recall and $F_1$-score (Table 16), complemented by confusion matrices (Table 17). In the unbalanced regime, accuracy is expected to be less informative due to skewed class proportions; therefore, the confusion

[2] `https://scikit-learn.org/stable/`.

matrices are used to explicitly inspect false positives and false negatives.

**Table 17:** Confusion matrices on ONOS under balanced and unbalanced training. Each entry is formatted as [TN FP; FN TP].

| Model | Balanced | Unbalanced |
|---|---|---|
| NB | $\begin{bmatrix} 222 & 0 \\ 2 & 220 \end{bmatrix}$ | $\begin{bmatrix} 999232 & 546 \\ 4 & 218 \end{bmatrix}$ |
| MLP | $\begin{bmatrix} 222 & 0 \\ 2 & 220 \end{bmatrix}$ | $\begin{bmatrix} 999774 & 4 \\ 10 & 212 \end{bmatrix}$ |
| AdaBoost | $\begin{bmatrix} 222 & 0 \\ 1 & 221 \end{bmatrix}$ | $\begin{bmatrix} 999777 & 1 \\ 1 & 221 \end{bmatrix}$ |
| DT | $\begin{bmatrix} 221 & 1 \\ 1 & 221 \end{bmatrix}$ | $\begin{bmatrix} 999777 & 1 \\ 1 & 221 \end{bmatrix}$ |
| RF | $\begin{bmatrix} 221 & 1 \\ 0 & 222 \end{bmatrix}$ | $\begin{bmatrix} 999777 & 1 \\ 1 & 221 \end{bmatrix}$ |
| KNN | $\begin{bmatrix} 222 & 0 \\ 6 & 216 \end{bmatrix}$ | $\begin{bmatrix} 999774 & 4 \\ 7 & 215 \end{bmatrix}$ |
| SVM | $\begin{bmatrix} 222 & 0 \\ 6 & 216 \end{bmatrix}$ | $\begin{bmatrix} 999776 & 2 \\ 46 & 176 \end{bmatrix}$ |

Overall, the balanced benchmark highlights the separation capacity of tree-based methods, with RF and AdaBoost providing the strongest results on ONOS. In the unbalanced setting, accuracy approaches 1.0 for most classifiers, but the confusion matrices show that error profiles remain heterogeneous: NB incurs a comparatively large number of false positives, whereas SVM exhibits a marked increase in false negatives (reduced recall on the `Malicious` class). For this reason, the unbalanced results are primarily interpreted through $F_1$/recall and confusion-matrix inspection rather than accuracy alone.

## 4.7 SHAP-based interpretability analysis

### 4.7.1 Motivation and methodology

While the performance metrics above indicate that several models can achieve high detection rates on KRONOS-SDN, understanding *why* a classifier flags a particular flow as malicious is essential for operational deployment and for validating that models rely on meaningful indicators rather than correlations correlated to bias introduced by the experimen-

tal setup. To this end, SHapley Additive exPlanations (SHAP) (Lundberg et al., 2017), a unified framework that attributes to each feature an importance value for individual predictions, was adopted.

For tree-based models such as RF and AdaBoost, the TreeSHAP formulation was adopted, which allows efficient exact computation of Shapley values. For the MLP, kernel-based SHAP is used on a stratified subsample of flows to control computational cost. For each machine and classifier, the beeswarm plots are graphed, where individual points represent flows and encode both the magnitude and sign of feature contributions to the prediction.

### 4.7.2 Global feature importance

Global SHAP provides support for the previous filter-based rankings as well. For the ONOS dataset, the most important contributions for both models are the features concerning idle time (`idle max`, `idle mean`) and metrics based on how long flows are active, and how quickly packets are forwarded.

For the Server VM, the most important contributing factors were features about the distribution of packet lengths (`fwd pkt len mean`, `fwd pkt len std`) and how fast packets arrive.

Some of the most important features for SHAP had relatively low Mutual Information (MI), F-tests, yet still have a significant contribution due to their participation in complex interactions between various features. This highlights the added value of model-based interpretability over purely univariate ranking.

### 4.7.3 Local explanations and attack-specific patterns

The SHAP explanations on KRONOS-SDN, a sample of which is shown in Figure 12, complemented by the graphs shown in the appendix C with Figures C.4, C.5 and C.6, suggest that model decisions are not dominated by a single correlation but instead arise from a more distributed set of cues spanning timing, transport-state dynamics and the dispersion of packet-size statistics. While several top-ranked variables still ex-

hibit interpretable separation patterns, such as specific inter-arrival time descriptors, duration-related indicators and direction-dependent packet statistics that tend to shift predictions toward the attack class when assuming atypical values, the overall attribution structure is less "one-dimensional" compared to the analysis conducted in Chapter 3.

A particularly informative element is the last bar in the global SHAP importance plots, i.e., the bucket aggregating the *sum of the remaining (non-displayed) features*. Across models, this aggregated contribution is widely distributed and often spans a substantial range of impact on the decision function, indicating that a considerable fraction of the predictive evidence is carried by the long tail of features rather than being concentrated in the top few. In other words, even if some features can present clear separations, the classifier still leverages many additional signals whose cumulative effect is significant; hence, its decision boundary is not reducible to a handful of easily manipulable variables.

From a security standpoint, this attribution spread is consistent with a dataset design and preprocessing effort aimed at avoiding trivial, correlations easy to bypass, e.g., purely monotonic rules on packet rates or minimum packet length. Therefore, for an adaptive intruder, there is an increased amount of complexity in order to have an attacker mimic benign behaviour on several partially informative characteristics such as timing, stateful TCP artifacts and distributional variability, instead of simply using some basic traffic shaping techniques (such as rate limiting, packet size variation, or variance among flows) to counteract a small number of predominant characteristics. The degree of practical difficulty associated with evasion has been significantly enhanced while at the same time, it has had little effect upon the efficacy of the attack; the ability to successfully evade detection will require coordinated management of a greater variety of heterogeneous types of observable behaviours.

## 4.8 Summary and methodological implications

This chapter has presented the methodology adopted for the design and analysis of the KRONOS-SDN dataset. The testbed architecture emulates

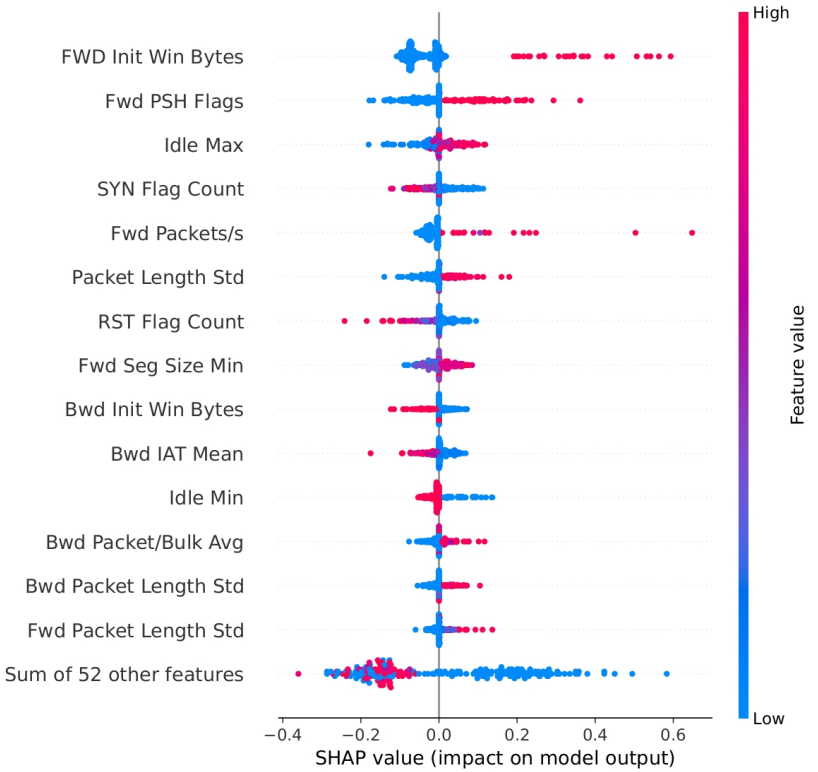



**Figure 12:** SHAP analysis for Support Vector Machine in Malicious-Normal traffic detection for ONOS VM.

a realistic SDN-enabled enterprise network with an ONOS controller, a router based on Linux kernel connecting different subnets, OVS-based switches and multiple host roles. By combining seven days of temporally coherent traffic with a manifold set of attack campaigns and by collecting both flow-level and host-level metrics, KRONOS-SDN addresses several gaps observed in existing SDN IDS benchmarks.

The descriptive statistics and feature analyses highlight the discriminative potential of idle-time, inter-arrival and packet-length features, while also revealing redundancy among certain groups of CICFlowMe-

ter attributes. Baseline machine learning experiments demonstrate that traditional classifiers can achieve high detection rates, especially when class imbalance is mitigated. SHAP-based interpretability further clarifies the feature contributions underlying model decisions, supporting both the validation of learned patterns and the design of more robust, context-aware IDS architectures.

These methodological elements underpin the subsequent part III, where the KRONOS-SDN dataset is leveraged to evaluate more advanced detection models and to investigate cross-layer fusion strategies that jointly exploit network flows and host utilization statistics.

## 4.9 Summary of KRONOS-SDN and Implications

In this chapter, an introduction to KRONOS-SDN (a realistic dataset for testing Intrusion Detection Systems) in Software-Defined Networks is provided with the rationale behind the design decisions that support its validity, reproducibility and real-world applicability. A deep description is provided about how the testbed architecture, the generation of network traffic and the creation of truth data were created in order to capture security phenomenon in SDNs under realistic operational conditions using controllable but operationally realistic conditions; such as heterogeneous benign workload, temporal coherency among attacks and different observation planes.

Beyond the dataset construction, the analyses presented throughout the chapter provide a set of empirical and methodological takeaways that directly shape the remainder of the thesis. First, the observed class imbalance and the diversity of attack dynamics motivate evaluation protocols that go beyond accuracy-centric summarizes and explicitly account for the error profile under realistic priors. Second, the exploratory feature analysis highlights that discriminative evidence is often concentrated in temporal and idle-time statistics and that redundancy is non-negligible, suggesting that model design and interpretability should be treated as first-order concerns rather than post-processing steps. Third, the avail-

ability of both traffic-derived representations and controller-side host telemetry establishes the prerequisites for cross-plane security studies, enabling the investigation of whether controller operating regimes can provide complementary context for traffic-based detection decisions.

In this context, *context-modulated thresholding* denotes a family of mechanisms in which the decision threshold of an anomaly detector is not fixed, but is adjusted at run time as a function of external signals that capture current operating conditions of the SDN controller and, more broadly, the control/data-plane regime. Concretely, KRONOS-SDN provides both flow-level features and controller/host-level telemetry, thereby enabling thresholds that depend, for instance, on controller CPU load, memory utilization, or control-plane activity indicators (e.g., flow-rule handling intensity) rather than on a static global value. This perspective motivates the adoption of adaptive sensitivity policies that can reduce false alarms during legitimate workload fluctuations while preserving responsiveness under suspicious controller-side regimes.

Taken together, these elements position KRONOS-SDN not only as a data contribution, but as an enabling substrate for systematic and deployment-oriented IDS research in SDN. These considerations motivate the experimental progression of the thesis. The next chapter consolidates the methodological foundations for ML-based IDS in SDN, with emphasis on deep, sliding-window anomaly detection and threshold calibration. Chapter 6 then instantiates a reproducible window- based pipeline on KRONOS-SDN to compare representative deep autoencoder families under homogeneous training and scoring conditions and to quantify both detection performance and deployability-relevant costs. Finally, the multi-plane nature of the data collected will allow to study how context-dependent changes to the sensitivity of the detector can be made by the controller based on the telemetry available at each plane, thus allowing us to develop the theoretical basis for moving beyond a baseline of traffic-only anomaly detection and toward a more robust, cross-domain approach to network anomaly detection.

# Part III

# New Approaches for Machine-Learning-Based Intrusion Detection Systems

# Chapter 5

# Background and Literature Review – ML-based IDS

Building on the methodological foundations established in Chapter 4, this part of the thesis transitions from the design and characterization of KRONOS-SDN to the investigation of learning-based intrusion detection methodologies, with specific attention to SDN environments. The exploratory analyses conducted on KRONOS-SDN highlight recurring empirical and methodological factors that are central to model design and evaluation in practice, including non-negligible feature redundancy, the prominent role of temporal and idle-time statistics, the operational impact of class imbalance and the practical value of post-hoc interpretability. These observations motivate the need for a structured discussion of ML-based IDS principles and for a critical review of the design choices that recur in state-of-the-art SDN IDS solutions.

In operational settings, intrusion detection must account not only for predictive performance but also for how decisions are produced and used. SDN has increased the need for these attributes as it has introduced programmable control logic and a security critical control plane; telemetry can now be richer and centralized to the controller, but this increases the possibility of both the controller being an additional bottleneck and attack target. As such, evaluations of IDS systems designed

around SDNs often evaluate how well they satisfy detection requirements while considering timing requirements, computational overheads and closed loop response stability. The above factors were used to create the experimental protocol that will be adopted in Chapter 6 where the first phase established a traffic-only sliding window-based deep anomaly detection baseline for KRONOS-SDN. In the second phase, a cross-plane coupling is created through a deployment oriented method in which the controller-based telemetry changes the sensitivity of the traffic detector at runtime.

Intrusion Detection Systems (IDSs) constitute a core component of modern cyber-defense. They monitor hosts and networks to raise alerts when observed behaviour either matches known malicious patterns or deviates from an expected baseline. Over time, IDS research has produced a broad spectrum of approaches, ranging from signature and rule-based engines to statistical detectors, data-driven machine learning (ML) models and hybrid multi-stage architectures. This chapter consolidates the concepts and design dimensions required to frame *ML-based* intrusion detection and it reviews the SDN-focused literature with emphasis on how SDN programmability and architectural separation reshape telemetry collection, detector placement and response coupling.

Consistent with the scope of this thesis, the discussion of non-ML IDS paradigms is intentionally not comprehensive. Classical signature engines, protocol specifications and purely rule-based correlation remain operationally relevant; however, the scientific and experimental core of the thesis focuses on ML-based IDS pipelines, including interpretability and context-aware thresholding. Accordingly, the background section introduces non-ML IDS principles only to the extent needed to motivate the shift toward learning-based detection, while the literature review provides a deeper account of ML-based IDS approaches and reported outcomes. The literature review will be structured by SDN-specific design dimensions (placement, telemetry and reaction coupling) and by learning paradigms (deep window-based detection, supervised, semi-supervised, etc.) with an emphasis on methodological aspects that affect generality, replicability and deployment within real-time constraints.

The rest of this chapter is divided as follows: Section 5.1 defines IDS-related basic concepts and design dimensions. Section 5.2 provides a summary of performance metrics and the factors that need to be taken into consideration during the evaluation of IDS systems which use thresholded anomaly scores as detection output. Finally, Section 5.3 will describe various IDS approaches using machine learning in SDN environments and will identify recurring methodological patterns and open issues that motivate the experimental study presented in Chapter 6.

## 5.1 Background: IDS Concepts and Design Dimensions

At a high level, an IDS performs three steps: collects observations from an environment (e.g., a host, a subnet or a controller), transforms raw telemetry into a representation suitable for analysis and, finally, applies a detection logic to decide whether the observation is related to benign or intrusive traffic. Depending on the design, the IDS may also classify attack families, correlate alerts over time or trigger mitigation actions.

### 5.1.1 Data Sources and Deployment Location

A persistent axis in IDS taxonomies is the vantage point and the type of telemetry being inspected (Lazarevic et al., 2005; Hindy et al., 2018; Alkasassbeh et al., 2023). An intrusion could potentially leave a variety of traces that can be analyzed depending upon the position of the observer; therefore, the selection of a source of data is going to have an effect on how well an intrusion can be detected and also with respect to what restrictions there will be in terms of operation. There is a first, simple way to distinguish IDS designs, based on the type of data sources they observe and the viewpoint from which this data is observed.

A *Network-based* IDS (NIDS) observes data from a strategic location within a network (for example, a router, switch, firewall, or a dedicated tap). They analyze packets, aggregated flows or protocol-level records. This perspective is well suited to identifying communication-centric

threats, such as scanning campaigns, volumetric denial-of-service, command-and-control activity and lateral movement, because it captures interactions across multiple hosts and subnets. At the same time, NIDS visibility can be substantially reduced by pervasive encryption and may be stressed by high-throughput links, where deep inspection and fine-grained state tracking become costly.

*Host-based* IDS (HIDS), instead, run on endpoints or servers and leverage host-local telemetry such as system and application logs, audit trails, system calls, kernel events, file-system activity and process-level traces. This vantage point is particularly effective for detecting compromise indicators that might not be directly observable on the network. For instance, local privilege escalation, persistence mechanisms or anomalous process behaviour. The operational trade-off is that HIDS require deployment and management across many endpoints, must handle potentially sensitive host data and can be exposed to tampering if an attacker gains sufficient privileges on the monitored machine.

Finally, *Integrated* IDS designs combine host- and network-level visibility, correlating signals from heterogeneous sensors (often through a central analysis layer comparable to SIEM-style aggregation). By fusing multiple perspectives, hybrid systems can improve detection coverage and provide richer context for triage and response. However, this comes with additional engineering complexity, including synchronization of data streams, consistent time alignment and the definition of reliable correlation logic across sources with different noise characteristics and sampling granularities.

Beyond this basic split, specialized detectors target particular protocols or services (e.g., DNS tunneling, web-layer attacks, ICS/SCADA command anomalies), exploiting domain semantics. At scale, collaborative and distributed IDS architectures exchange alerts or compact summaries across vantage points to detect attacks that are stealthy locally but suspicious globally (Vasilomanolakis et al., 2015).

### 5.1.2 Detection Principles: From Rules to Models

An additional fundamental dimension of IDS is *how* an IDS identifies benign versus malicious activities as reported by (Gyanchandani et al., 2012; Khraisat et al., 2019). Signature-based (misuse) IDSs identify malicious behaviour when it matches a pre-existing pattern. The signature-based approach has a high operational value to provide precise, understandable alerts for well-defined attacks. However, due to the ongoing nature of malware, their operation and maintenance is required continuously in order to be effective and they are ineffective in identifying entirely new types of attacks as well as attacks that have been heavily obfuscated through techniques such as encryption and mimicking protocols.

A second paradigm is *ML-based classification* (often referred to as *knowledge-based* or *supervised* detection), where an IDS learns a discriminative mapping from telemetry features to attack categories (or to a benign/malicious decision as well as multiclass one) using labeled data. In contrast to signature matching, this approach can generalize beyond exact patterns by exploiting statistical regularities in feature space and it naturally supports multi-class outputs and calibrated confidence scores. Its main limitation is that performance depends on the representativeness and freshness of labels: class imbalance, dataset shift and domain mismatch may degrade generalization, while high-quality labelling is expensive and may be unavailable in operational environments. As a consequence, supervised ML-based IDSs are often effective within well characterized domains but can require periodic retraining and careful validation to remain reliable under evolving traffic conditions.

Anomaly-based systems, by contrast, model normal behaviour and flag statistically or semantically unlikely deviations. This paradigm is attractive because it can, in principle, expose unknown or evolving threats, but it must contend with concept drift, unstable baselines and the risk of high false-positive rates if ”normal” behaviour is not adequately captured (Hindy et al., 2018). Hybrid IDS designs combine multiple principles. For instance, anomaly detectors can triage suspicious regions of telemetry that are then examined by more precise signature modules or

outputs can be fused via correlation rules or meta-classifiers (Alkasassbeh et al., 2023).

### 5.1.3 Learning Paradigms and Adaptivity

Within learning-based IDS, the learning paradigm determines what supervision is available and how the detector adapts over time. Surveys commonly distinguish supervised, unsupervised, semi-supervised and online/incremental regimes (H. Liu et al., 2019; Aldweesh et al., 2020).

From a machine learning perspective, IDS designs can be distinguished by the kind of supervision and feedback they can leverage during training and operation. In *supervised* settings, the detector learns a decision function from labeled examples of benign and malicious traffic. This formulation tends to yield strong performance in controlled benchmarks, but it is tightly coupled to the availability and quality of labels, as well as to the extent to which the training set covers the threat space that will be encountered at deployment time.

In *unsupervised* detection, training data are assumed to be predominantly benign and the model focuses on identifying rare or atypical observations without relying on explicit attack labels. In practice, this shifts the design effort toward selecting robust anomaly scores and calibrating decision thresholds, while also coping with non-stationarity in benign workloads, which can otherwise inflate false alarms.

*Semi-supervised* methods fall between supervised and unsupervised methods. Like unsupervised methods, they utilize large amounts of benign or unlabeled traffic to develop feature representations or baselines; however, like supervised methods, they use a small amount of labeled traffic to finetune the thresholds used by the model, to better define the boundaries around each class, or to help guide the development of representations. Semi-supervised methods offer cost savings compared to supervised methods, while improving the applicability of machine learning to real-world problems; however, this comes with challenges related to validating performance for semi-supervised methods that rely on only a few labeled examples and which may not have been trained to repre-

sent the diverse set of possible attacks.

Finally, *online* or *incremental* learning explicitly targets long-running deployments by continuously updating the detector as new data arrive. This increases adaptability to evolving traffic patterns and changing adversarial tactics, but introduces additional operational constraints: updates must be stable, verifiable and resilient to adversarial manipulation (e.g., poisoning) and the system must balance responsiveness with the risk of drifting away from a trustworthy baseline.

The importance of adaptability has grown, such that it is now considered one of the key requirements for deploying IDS systems in practice: IDS systems should receive feedback from false alarms, missed detections and changes in normal workload (Alkasassbeh et al., 2023); this leads to many of the themes that we will see repeatedly in this chapter: calibration of thresholds, detection of drift and the operational cost of updating the model.

### 5.1.4 Detection Timing, Response and Architecture

The detection may occur in real-time, near real-time (through buffering or through time window aggregation) or off-line for forensic purposes and/or retraining. Near real-time means that the detection system produces an alert within seconds since the moment when the attack begins (i.e., typically between 1 second and 10 seconds), therefore allowing for mitigation to begin without experiencing significant delays as a result of the underlying traffic dynamics.

Response semantics span passive alerting to automated prevention (IDPS), where detection is directly tied to actions such as blocking, rate limiting or re-routing traffic. Architecturally, IDSs may be centralised (global analysis engine), hierarchical (local pre-processing with upstream correlation) or fully distributed with peer-to-peer alert exchange (Vasilomanolakis et al., 2015). These choices govern scalability, latency and the impact of attacks against the IDS itself.

## 5.2 Performance Evaluation of IDS

IDS performance is typically summarised through confusion-matrix derived metrics. In the binary case, true positives ($TP$), true negatives ($TN$), false positives ($FP$) and false negatives ($FN$) yield:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.2}$$

$$\text{Recall (TPR)} = \frac{TP}{TP + FN} \tag{5.3}$$

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.4}$$

In addition, Receiver Operating Characteristic (ROC) curve plots the *true positive rate* (TPR) against the *false positive rate* (FPR) as the decision threshold is swept over the full range of scores, thus visualizing the trade-off between sensitivity and false alarms. The Area Under the ROC Curve (AUC) corresponds to the integral of this curve and can be interpreted as the probability that a randomly chosen positive sample receives a higher score than a randomly chosen negative sample; $AUC = 0.5$ indicates chance-level discrimination, whereas $AUC = 1$ denotes perfect separability. Per class ROC Curves and AUC Values are typically created through the use of One-Versus-Rest decomposition in multi-class settings.

ROC curves and AUC provide threshold-independent summaries when the IDS emits a continuous score (e.g., a posterior probability, a confidence value or an anomaly score).

Although these metrics are commonly used, additional practical metrics for measuring the performance of an SDN deployment have been shown to be of high value: detection delay, controller overhead, additional control plane Traffic resulting from telemetry collection and the stability of closed loop responses. The reporting of these metrics is inconsistent throughout the literature and represents a key methodology concern for the design of near-real-time IDS systems.

## 5.3 Literature Review: ML-based IDS in Software-Defined Networks

The general IDS taxonomy introduced in Section 5.1 and the evaluation criteria discussed in Section 5.2 provide a necessary baseline, but they are not sufficient to characterize intrusion detection in Software-Defined Networks. Indeed, SDN alters the *operational context* in which detection is performed: beyond deciding whether an event is malicious, an SDN-oriented IDS must also cope with the architectural separation of functions, the programmability of control logic and the tight coupling between detection and automated response. Consequently, classical performance metrics (e.g., precision, recall, $F_1$, AUC-ROC) must often be interpreted together with system-level indicators such as detection latency, control-plane overhead and mitigation stability, because an IDS that is accurate yet slow or disruptive can be impractical in a programmable network.

SDN separates the control plane from the data plane and exposes programmable interfaces for network management (Open Networking Foundation, 2014b). This programmability enables fine-grained policy enforcement and rapid reconfiguration, but it also expands the attack surface: the controller, the southbound and northbound interfaces and the interaction between distributed switches and centralized control logic become security-critical components (Diego Kreutz et al., 2015; Scott-Hayward et al., 2016). As a consequence, IDS designs for SDN must address both classical traffic threats and SDN-specific attacks such as controller-targeted DDoS, control- channel manipulation and flow-table exhaustion (Batool et al., 2024).

A major outcome of the last decade of research is that SDN is not merely another deployment environment: it changes *what* can be measured (e.g., flow counters, controller events), *where* detection can be placed (controller, switch or auxiliary analytics) and *how* responses can be enforced (programmable mitigation). Surveys of SDN security consistently highlight both the opportunity of leveraging global visibility and the risk of overloading or attacking the controller (Hande et al., 2020; Chica et al.,

2020).

These characteristics drive SDN-IDS architecture that specifically trade-off between detection performance, the safety of the control plane and the responsiveness of operations and, therefore, why datasets and experiment protocols for SDN-IDS should document both the trace of an attack and its label as well as the viewpoint from where the detection is to take place, the level of detail in the telemetry and the time constraint on when the detection is to occur.

SDN IDS designs generally have at least three practical design restrictions:

- telemetry must continually be collected and converted to machine readable format while being subject to reasonable overhead;
- detection results must be reported to the controller in real-time or as close to real-time as possible and this may require use of windows/streaming techniques over using batch classification on individual records;
- the cost associated with both computation (the time required to extract features from raw traffic and the number of packets processed per second) and the cost of the control plane (the amount of processing time and packets processed per second) will impact overall network stability.

Therefore, in order to evaluate the effectiveness of SDN IDS solutions, it is important to provide not only predictive performance metrics, but also end-to-end cost and timing characteristics in order to determine whether the performance of the system provides sufficient accuracy for the actual operational costs of the system.

### 5.3.1 Design Axes Specific to SDN IDS

Two SDN-specific aspects recur across ML-based IDS proposals.

**Telemetry and representation.** Many SDN IDS adopt flow-level features derived from switch counters (bytes/packets/duration, TCP flags,

rates) or aggregate such features over time bins and entities (e.g., per source IP, per destination service). Some proposals extend the representation with controller logs or resource indicators, aiming at cross-plane visibility, although such designs remain less common in the broader SDN IDS literature (Sultana et al., 2019).

**Deployment and reaction coupling.** Controller-centric IDS designs can easily integrate detection and response: once suspicious behaviour is inferred, the controller can install mitigation rules (drop, rate limit, reroute) with low orchestration latency. Distributed alternatives place lightweight components near the data plane or as virtualized functions to improve scalability and resilience, at the cost of partial visibility and synchronization complexity (Jafarian et al., 2020). These trade-offs strongly influence which ML techniques are viable under real-time constraints.

### 5.3.2 Supervised ML for SDN Intrusion Detection

Most early ML-based SDN IDS approach detection as a supervised classification task inferring on flow metrics. Many traditional classifiers like decision tree, random forest, $k$NN, SVM, Naïve Bayes and ensemble methods were trained with labeled benign and malicious samples, mostly focusing on DDoS and saturation-type attacks since these types of attacks have clear signatures that show up in aggregate flow counters.

A representative line of work uses SVM on SDN flow features to separate benign from DDoS traffic, showing that even relatively lightweight classifiers can achieve strong discrimination in controlled testbeds (S. Li et al., 2018). Beyond linear separators, tree-based ensembles have been adopted to better capture non-linear feature interactions; gradient boosting and XGBoost-style models are frequently reported to improve robustness on heterogeneous traffic mixes, including SDN-based cloud settings (Chen et al., 2018). Across surveys, the recurring empirical message is that supervised learning often performs well on benchmark datasets and lab-generated SDN traces, but generalization across con-

trollers, topologies and workload regimes remains a central concern (Xie et al., 2019).

The limitations of Supervised Learning detailed above are especially notable within the context of SDNs; gathering sufficient amounts of representative, labeled data for use as a dataset is expensive; there could be attacks that are not represented or covered in the training data and at the time of testing a threat could have been different than when training was completed. Therefore, supervised models are typically integrated into larger pipeline approaches. For example, to improve multi-attack classification after an anomaly detector has identified potentially suspicious traffic.

### 5.3.3 Unsupervised and Semi-supervised Detection in SDN

To reduce the dependence on labels and improve sensitivity to unseen attacks, SDN IDS research has explored unsupervised and semi-supervised paradigms. Clustering-based detectors (e.g., $k$-means, density-based methods, SOM) learn structures of benign traffic and treat outliers as anomalies; their appeal lies in conceptual simplicity and low training requirements, but they often require careful feature scaling and threshold calibration to keep false alarms manageable (Jafarian et al., 2020).

Probabilistic Temporal Models (HMM-like techniques), for instance, have also been used to model sequential dependencies in control-plane or aggregate flow data. Their argument is that low rate or multistage attacks could be identified through temporal anomalies, rather than snapshot anomalies. These types of probabilistic temporal models emphasize the use of temporal modeling as a means of addressing some of the shortcomings of stateless classification; however, their success will depend heavily upon the stationarity of the normal regime.

Unsupervised deep learning extends this idea by learning compact representations of benign behaviour. Dawoud *et al.* propose an SDN anomaly detection scheme that combines stacked autoencoders with clustering, leveraging non-linear representation learning to separate normal operation patterns from deviations (Dawoud et al., 2019). The majority

of semi-supervised approaches use some small amount of labeled attack data to set thresholds or refine the decision boundary; these approaches are designed to be more practical for use while maintaining relatively low cost of obtaining labels.

In summary, both unsupervised and semi-supervised approaches have been reported to improve the likelihood of detecting previously unknown attacks by the SDN community, however, both have significantly increased operational challenges including; drift, stability of thresholds and ability to understand alerts become first order concerns.

### 5.3.4 Deep Learning-Based IDS in SDN

Deep learning (DL) has been adopted in SDN IDS to learn hierarchical representations and capture complex non-linear patterns that may be poorly approximated by handcrafted features alone (Xin et al., 2018). Systematic reviews of DL-based SDN security show a strong emphasis on DDoS detection, reflecting both the prevalence of DDoS as an SDN threat and the relative ease of generating such traffic for evaluation (Ali et al., 2023).

#### DNNs and autoencoder-assisted classification

Niyaz *et al.* propose a DL-based DDoS detector for SDN where a Deep Neural Network (DNN) autoencoder learns compressed representations of flow features, which are then classified via a supervised module; the reported outcome is improved discriminative performance compared to shallow baselines under their experimental setting (Niyaz, W. Sun, and A. Javaid, 2017). Similar designs are repeatedly observed in SDN IDS prototypes: unsupervised representation learning is used to stabilize features, followed by supervised classification for decision-making.

#### CNN-based models

Convolutional Neural Networks (CNNs) have been employed to capture local patterns in structured representations of traffic statistics or packet-derived features. Haider *et al.* report that CNN ensembles can improve

robustness for SDN DDoS detection by combining multiple convolutional views of the input (Haider et al., 2020). In IoT-oriented SDN contexts, near-real-time CNN-based detectors are often presented as meeting bounded-latency requirements while achieving strong detection accuracy on the evaluated traces (Assis et al., 2020).

#### RNN/GRU/LSTM temporal modeling

Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) models explicitly model sequences, which is particularly relevant when attacks evolve over time rather than appearing as isolated flow outliers. Assis *et al.* show that Gathed Recurrent Unit (GRU)-based models can capture temporal evolution of SDN traffic features to detect attack behaviours that are difficult to identify from single snapshots (Assis et al., 2021). The broader empirical trend is that temporal models can reduce missed detections for bursty or slowly evolving attacks, at the cost of higher complexity and the need for disciplined windowing and evaluation protocols.

#### Ensembles and lightweight online detectors

Ensemble strategies aim to improve stability and reduce variance by aggregating multiple models or feature subsets. The online ensemble-of-autoencoders design of Kitsune has inspired SDN-oriented adaptations where streaming detection and computational efficiency are emphasized (Mirsky et al., 2018). Griffin-style ensemble approaches similarly illustrate how multiple autoencoders can be combined with downstream classifiers to produce robust detection pipelines (Griffin et al., 2020).

Despite the promising detection outcomes reported across these DL families, two recurrent patterns emerge from the SDN literature. First, many prototypes remain largely framed as *closed-set* supervised classifiers, often optimized for DDoS scenarios, whose validity depends on the representativeness of labeled traces and on the stability of the traffic distribution used for training and testing (Ali et al., 2023).

Second, SDN imposes system-level constraints that intertwine learning quality with operational feasibility: telemetry is produced continuously, decisions are expected under bounded latency and excessive computation or control-plane interaction may itself become a risk factor (e.g., by amplifying controller load).

These findings suggest a transition to a new paradigm of *streaming* approaches in SDN intrusion detection to address both:

- naturally match how SDN measurements are collected;
- can provide detection even when labels are incomplete or when attacks appear as previously unseen variants.

A particular influential approach to real-time anomaly detection in SDN is anomaly-based deep learning using sliding time windows. This approach uses the same representations (autoencoders, temporal networks etc.) but frames the problem as defining what "normal" looks like on a continuous stream of telemetry data, with significant deviations being flagged as anomalies; this represents an operational compromise between detection capabilities and labelling requirements (Mirsky et al., 2018; Xin et al., 2018).

**Synthesis: Architectures for empirical comparison**

Given the diversity of deep architectures proposed for SDN IDS, the empirical evaluation in Chapter 6 focuses on four representative autoencoder families:

- **CNN-AE**: convolutional autoencoders capturing local temporal patterns with relatively low inference cost;
- **LSTM-AE**: recurrent encoder-decoder models explicitly modelling sequence order and temporal dependencies;
- **Tr-AE**: transformer-based autoencoders leveraging self-attention to capture long-range dependencies and parallelize computation on GPUs;

- **MLP-AE**: feed-forward autoencoders providing a minimal, CPU-friendly baseline and enabling architectural reuse across traffic and host telemetry.

This selection balances representativeness (convolutional, recurrent, attention-based, feed-forward) with experimental tractability in a controlled comparison on KRONOS-SDN.

Given its central role in the experimental contribution of this thesis, sliding-window *anomaly-based* deep learning is not treated merely as one additional DL variant within the SDN IDS landscape; instead, it is developed in a dedicated section to formalize the windowing abstraction, thresholding and deployment constraints that specifically govern near-real-time operation and that underpin the empirical study in Chapter 6.

## 5.4 Anomaly-based Deep Learning on Sliding Time Windows

The sliding-window formulation treats SDN telemetry as a multi- dimensional time series, not as individual flow records. Thus, instead of evaluating each record independently, the detector evaluates multiple records over a sliding window, determines a window-level aggregation and assigns a score to this one. This method is well-suited to SDN operational needs; it provides for continuous processing, it generates decisions at a defined level of temporal resolution and it allows the user to trade-off response-time, level of noise-reduction and computational costs.

More recent surveys on time-series anomaly detection have framed windowing as a practical connection between streaming limitations (e.g., compute and latency) and learning from temporal representations, particularly for multivariate telemetry that may contain anomalies which appear to be weak in local data, but are consistent across short and medium term histories (Zamanzadeh Darban et al., 2024a; F. Wang et al., 2025).

The main idea here is that many types of attacks present characteristic temporal patterns (bursts, ramps, sustained plateaus, low rate periodic-

ity), that are not easily identified by examining single snapshot values, but would emerge when viewing a history of telemetry values of a short period of time. In this way, the sliding window model serves both as a modeling decision (allowing reconstruction- or prediction-based anomaly objectives) and as an evaluation point: defining the time scale of detection, the latency of alert generation and the degree of temporal context being used (which influences the evaluation of the performance metrics presented in Section 5.2).

### 5.4.1 Representation and problem formulation

Let $\{\mathbf{x}_e(t)\}_{t=1}^{T_e}$ be a time-indexed stream of feature vectors describing SDN telemetry for a monitored entity $e$ at tick resolution $\Delta t$, where each vector $\mathbf{x}_e(t) \in \mathbb{R}^d$ contains $d$ telemetry features extracted for the corresponding time bin. The monitored entity may correspond, depending on the adopted configuration, to a host, a source-destination pair or a more fine-grained traffic identifier, for example a five-tuple characterized by source and destination IP, ports and protocol.

Given a window length $L$, the detector maintains an independent temporal buffer for each entity. During the initial warm-up phase, no complete input is emitted until the buffer has accumulated $L$ observations. Once the buffer is full, a new window is generated whenever a new tick is available for that entity. Each runtime window ending at tick $t$ is therefore defined as:

$$\mathbf{X}_{e,t} = [\mathbf{x}_e(t-L+1), \mathbf{x}_e(t-L+2), \ldots, \mathbf{x}_e(t)] \in \mathbb{R}^{L\times d}, \qquad t \geq L \quad (5.5)$$

Here, $L$ denotes the number of consecutive ticks included in each window, $d$ is the number of telemetry features per tick and $\Delta t$ determines the temporal granularity at which observations are emitted. Thus, after the warm-up phase, runtime windows advance by one emitted tick for each active entity.

Models then output anomaly scores in unsupervised and semi-supervised settings. The SDN literature frequently motivates windowing as a compromise between responsiveness and context: shorter windows reduce

detection delay and improve coverage of short-lived behaviours, whereas longer windows increase temporal coverage but introduce a longer warm-up phase, larger per-entity state and higher computational cost (Ali et al., 2023).

### 5.4.2 Architectural patterns and reported outcomes

There have been a number of implementations of sliding-window anomaly detection paradigm using various types of Recurring Deep Architectures, that are detailed below.

**Reconstruction-based sequence autoencoders.** Autoencoders typically utilize Recurrent Neural Network (LSTM/GRU) encoders and decoders to train to reconstruct "normal" sliding windows; reconstruction errors can be used as a form of anomaly scoring. Autoencoder-based pipelines have been frequently referenced as a form of baseline for anomaly detection in SDNs as they do not rely on labelling attacks and can be implemented as continuous scorers within streaming pipelines (Dawoud et al., 2019).

**Prediction-based sequence models.** Instead of reconstructing the current window, predictive models generate a future value (either a future window summary or a future window value) and use the difference between predicted values and actual values as the anomaly score. The GRU/LSTM detector is commonly applied in SDNs, as it captures normal behaviour and reveals minute timing anomalies that are indicative of stealthy attacks (Assis et al., 2021).

**Hybrid CNN–RNN and attention-based designs.** Hybrid models consist of convolutional blocks combined with Recurrent Layers or Attention Layers (to catch temporal dependency). Hybrid models have been demonstrated to provide near-real-time performance and to effectively detect bursty attacks with low computational overhead (Assis et al., 2020).

Recently, Transformer-based Self-Attention Models have become increasingly popular in Time Series Anomaly Detection due to their ability to model long-term relationships without the need for recurrent unrolling and enable fully parallel training and inference. TSAD Surveys and representative transformer-based detectors demonstrate the growing interest in Transformer-based semi-supervised anomaly detection, but also highlight their limitations (in terms of required computing power and data) (Zamanzadeh Darban et al., 2024a; F. Wang et al., 2025).

**Ensembles and multi-scale windows.** Multi-scale methods utilize multiple detector methods operating over different window sizes to capture both short-burst and long-term trends and then aggregate the scores from each method or scale. The concept of multi-scale methods is similar to that of lightweight ensemble methods for detectors, such as Kitsune, which emphasize modularity and streaming operation (Mirsky et al., 2018).

### 5.4.3 Thresholding and context-aware severity

Continuous output values from Anomaly Detectors must be converted into discrete decisions (benign vs. anomalous), or ordered severity levels. The conversion of continuous values to discrete decisions (or ordered severity) is accomplished through thresholding mechanisms.

**Terminology.** For clarity and consistency, the following terminology is adopted throughout this thesis:

- *reconstruction error* refers to the raw discrepancy between an autoencoder input and its reconstruction, quantified through a pointwise loss such as MSE or MAE;
- *anomaly score* denotes a scalar statistic derived from reconstruction error and used as the main quantity for decision-making, for instance by aggregating errors over a window or by retaining the error associated with the most recent time step;

- a *static threshold* indicates a fixed decision boundary applied to anomaly scores, typically calibrated as a high quantile of the benign score distribution (e.g., $q_{99.5}$) in order to bound the false-positive rate;
- a *context-modulated threshold* denotes a time-varying decision boundary whose value is selected as a function of contextual variables capturing the operating regime (e.g., control-plane workload), with the aim of adapting detector sensitivity without retraining.

In this terminology, *threshold* designates the decision boundary applied to the anomaly score, whereas *modulation* refers to the systematic adaptation of that boundary based on context.

A common practice is quantile-based calibration on benign scores, selecting high quantiles to bound the false-positive rate (Mirsky et al., 2018). Recent studies refine this approach in three main directions: adopting multiple quantiles to implement multi-level severity mappings (Aoudi et al., 2021a); conditioning thresholds on context variables such as time-of-day, workload regimes, or device roles (Clausen et al., 2021; Ortega-Fernández et al., 2022; Borgioli et al., 2024); finally, adapting thresholds at the entity level to mitigate biases induced by majority traffic patterns and heterogeneous baseline volumes (Clausen et al., 2021; Borgioli et al., 2024; S. H. Sun et al., 2024).

In SDN Environments, as the Controller consumes Severity Signals to Select Mitigation Actions based on their relative Operational Cost (logging, rate limiting or blocking). As such, Chapter 6 builds upon this body of work with an approach to Threshold Modulation for Stabilizing Decisions across Heterogeneous Operating Regimes using Cross-Plane and Cross-Domain Context.

### 5.4.4 Deployment considerations

Window-based deep anomaly detection in an SDN architecture is also associated with challenges of feature extraction in stream and efficient use of resources. Since the maximum amount of information can be gathered if all feature calculations occur at the controller, it is possible that

the controller will become a bottleneck for traffic flow. For this reason many SDN IDS architectures suggest either using additional analytical components to aggregate telemetry, or placing pre-processing logic close to the data plane (Hande et al., 2020). In terms of inference time, CNN- and GRU-based models have been shown to be suitable for near-real-time operations on commodity hardware, while larger attention models and ensembles may require batching strategies or accelerators to meet their latency budget (Ali et al., 2023).

Finally, because of concept drift, there is a requirement for mechanisms that can facilitate re-calibration (i.e., retraining/semi-supervised fine-tuning using rolling quantiles or periodically). The literature has acknowledged this requirement, however, rigorous, reproducible evaluations of online adaptation over extended periods in realistic SDN environments remains relatively limited.

### 5.4.5 Datasets, Evaluation Methodology and Deployment Trade-offs

Dataset availability restricts ML-based SDN IDS performance evaluations. Most studies use general-purpose IDS datasets (e.g., KDD'99, NSL-KDD, UNSW-NB15, CIC-IDS2017, CSE-CIC-IDS2018) that do not include SDN specific artefacts such as those created between a controller and switch or within the control plane, as was previously noted in Section 3.1. Additionally, most SDN-specific datasets and testbeds focus primarily on very few types of attacks (most notably DDoS) and will not adequately capture the benign variability and multi-stage nature of operational networks (Ring et al., 2019).

Additionally, this lack of SDN datasets affects methodologies in several indirect ways. The typical method used to evaluate window-based detection methods involves creating retrospective windows from existing data traces. However, these retrospective windows may not accurately represent the limitations imposed by real-time streams or the additional overhead of the controller. In addition to reporting the typical measures of accuracy and F1-score, resource based measures (such

as CPU usage and memory utilization of controllers, increased control-plane traffic, etc.) are typically not reported with similar frequency, although they are key factors in determining the feasibility of deploying an SDN.

As a result of the above, the decision space for evaluating SDN IDS is multidimensional. While centralized designs offer greater context and better potential for bottlenecks and single points of failure, distributed designs may have better scalability but lose some ability to identify relationships across the network. Sliding window deep anomaly detectors may provide greater temporal resolution to detect anomalies over time, however, larger windows and more complex models may introduce increased latency and computational costs.

### 5.4.6 Open Challenges and Research Directions

There exists a number of unresolved issues in the SDN IDS literature, which are particularly critical to the design of ML-based and window-based detectors, since the performance of these detectors is highly dependent upon their operation within given operational constraints (availability of telemetry data, latency budget and safety of the controller).

**Realistic SDN datasets with rich ground truth.** One of the main bottlenecks in the SDN IDS literature is the lack of realistic SDN datasets that capture realistic SDN deployments, a variety of different traffic mixes, interactions between the control-plane and the data-plane and a variety of different attack-types, while providing temporally consistent labels and sufficient documentation for reproducibility. This is particularly important when evaluating streaming and window-based detection, where time alignment and the fidelity of the ground-truth labels are critical to both measuring the accuracy of the detector and its latency (Janabi et al., 2024; Ali et al., 2023).

**Cross-plane and cross-domain data fusion.** Most existing SDN IDS prototypes use only a single observation plane (the flow-level traffic fea-

tures), however most relevant security evidence will be located in multiple planes and domains: an attack may cause only slight changes in the data-plane traffic, however leave more distinct marks in control-plane events (such as rule-install bursts, topology updates) or as resource and process level effects on hosts. Therefore, the literature has increasingly pointed to the necessity of *cross-plane* fusion (of data-plane and control-plane telemetry data) and *cross-domain* fusion (of network measurements with host/application indicators) to improve the robustness of IDS and decrease the ambiguity between benign anomalies and malicious behaviour (Hande et al., 2020; Chica et al., 2020; Janabi et al., 2024). However, the practical implementation of such fusion under the streaming constraints (including, e.g., heterogeneous sampling rates, missing modalities, temporal misalignment and increased operational overhead) is currently difficult.

**Online learning, drift and operational stability.** While sliding- window scoring naturally matches streaming telemetry, robust procedures for continuous model updating remain an open problem. In operational networks, traffic regimes drift over time and concept drift may be adversarially induced; consequently, update strategies must preserve security guarantees and avoid silently degrading detection through contaminated baselines. Recent IDS-focused surveys on concept drift and feature dynamics emphasize that drift handling is not a secondary optimization, but a deployment requirement that affects threshold stability, retraining cadence and the reproducibility of long-running evaluations (Shyaa et al., 2024).

**Explainability and operator trust.** Deep sequence models and high-capacity detectors are often opaque. Explainability is therefore increasingly treated as a practical requirement in IDS pipelines, not only to support alert triage and root-cause analysis, but also to increase operator trust and facilitate iterative calibration under evolving workloads. Recent reviews of explainable AI in IDS highlight both the popularity of post-hoc attribution methods and the still-open challenges of producing

stable, comparable explanations under streaming constraints and across heterogeneous entities (AL et al., 2025; Mohale et al., 2025).

**Robustness to adversarial manipulation.** ML-based IDS can be attacked through evasion and poisoning. Temporal models introduce additional vulnerabilities: adversaries may shape traffic over time to mimic benign dynamics, or gradually shift baselines to suppress alarms. Recent surveys specifically focused on adversarial machine learning against IDS document a broad taxonomy of attack strategies and defenses, while broader meta-surveys consolidate the fast-evolving landscape of adversarial threats against modern learning systems (Alotaibi et al., 2023; Pawlicki et al., 2025).

**Closed-loop integration with SDN control.** SDN enables fast mitigation, but coupling detection outputs to controller actions creates feedback loops that must remain stable and safe. Mapping severity levels to mitigation policies (rate limiting, re-routing, isolation) requires careful orchestration, particularly under multi-controller or multi-domain deployments, where policy conflicts and cascading effects may arise (Latah et al., 2018).

In summary, ML-based IDS in SDN have evolved from classical supervised classifiers operating on flow statistics to deep, window-based detectors that explicitly model temporal structure. The literature reports promising outcomes, often under DDoS-centric threat models, yet repeatedly highlights practical barriers: dataset realism and ground-truth richness, drift and operational stability, the still-fragmented treatment of cross-plane/cross-domain fusion, explainability, robustness and deployability under controller constraints.

These limitations directly shape the focus of this thesis. First, the aim was to address the dataset gap by introducing KRONOS-SDN as a realistic SDN benchmark with temporally coherent traffic, heterogeneous attack campaigns and rich ground truth. Second, cross-plane and cross-domain data fusion strategies are investigated with the joint exploitation of network-flow statistics and host utilization telemetry to improve ro-

bustness under near-real-time constraints. Third, post-hoc interpretability is investigated to validate learned signals and support analyst trust.

## Bridge to Empirical Evaluation

The current chapter has reviewed the field of IDS based on ML, focusing on constraints and open challenges specific to SDNs. It specifically emphasized that while reported accuracy metrics (i.e., Precision, Recall, $F_1$, AUC-ROC) need to be evaluated, they should be considered in conjunction with operational parameters (i.e., detection latency, controller overhead and resource usage), since they are needed for evaluating IDS performance in actual deployment scenarios.

To address these gaps, Chapter 6 introduces a systematic empirical evaluation on the KRONOS-SDN dataset. The study instantiates a unified sliding-window anomaly-detection pipeline operating on flow-based traffic and compares four representative autoencoder families (CNN-AE, LSTM-AE, Tr-AE and MLP-AE) under homogeneous training and scoring conditions. Beyond reporting detection effectiveness, the evaluation explicitly quantifies deployability-relevant costs by measuring feature-extraction overhead, inference latency and throughput and the computational footprint in terms of CPU/RAM (and GPU, when applicable) utilization. Finally, the chapter proposes a cross-plane, context-modulated thresholding mechanism that couples data-plane traffic anomaly scores with controller-side telemetry to adjust detector sensitivity at runtime. Overall, this design translates the methodological considerations of the present chapter into a reproducible experimental protocol on a realistic SDN benchmark.

# Chapter 6

# Deep learning-based anomaly detection on KRONOS-SDN

Based upon the gaps described within Chapter 5, there exists a need to move forward from survey level data to controlled, pragmatic and repeatable evaluations with respect to realistic SDN benchmark datasets described in Chapter 4. Sliding-window detectors are not simply being evaluated based upon their ability to detect anomalies in operational networks; they must also meet the requirements with respect to near-real-time feasibility. Consequently, alerting behaviour depends jointly on error trade-offs (false alarms versus missed attacks), latency and throughput constraints, and the computational footprint of the end-to-end pipeline. As a result, each type of sliding window detector has a unique computational footprint associated with the entire pipeline. A controlled evaluation framework was created using KRONOS-SDN, and it includes a multi day campaign and a common telemetry stack.

Chapter follows a two-stage progression. First, a traffic-only baseline is instantiated by translating the sliding-window formulation into an end-to-end experimental design (Section 6.1) and formalizing the study objectives through guiding research questions (Section 6.2). Packet cap-

tures are converted into model-ready $L \times d$ tensors through a streaming windowing pipeline (Section 6.3) and four representative autoencoder families are compared under homogeneous training, calibration and window-level decision semantics: CNN-AE, LSTM-AE, Transformer-AE and MLP-AE (Sections 6.4 and 6.5).

Window-level scoring and labelling are fixed and reported explicitly to ensure architectural comparability (Section 6.6). Performance and efficiency metrics are then defined as a self-contained reference (Section 6.7) and used to conduct a controlled benchmark evaluation on KRONOS-SDN, resulting in the selection of a single reference detector (MLP-AE) for subsequent analyses (Section 6.8). Second, building on the traffic-only baseline, the chapter moves toward cross-plane anomaly detection by introducing a coupling mechanism in which controller-side host telemetry collected via `collectd` is used to infer an operating-regime signal that modulates the traffic-detector threshold at runtime (Subsection 6.8.8). This second stage tests whether cross-plane context can improve robustness without changing the core traffic detector, providing deployment-oriented evidence on how lightweight fusion primitives can stabilize anomaly decisions in realistic SDN environments.

## 6.1 From Literature to Experimental Design

As discussed in Subsection 5.4 of Chapter 5, deep anomaly detection over sliding windows has become a pragmatic design choice for near-real-time monitoring, because it transforms an unbounded stream into fixed-length tensors and enables decisions at a controlled temporal granularity. In the implementation used in this thesis, this granularity is jointly specified by the window length $L$ and the tick resolution $\Delta t$, which determines how often per-entity observations are produced from the packet stream. It should be emphasized that the notion of an entity was defined in general terms in Subsection 5.4.1 and will be operationally defined in Subsection 6.3 and has been applied consistently throughout this chapter. The formulation used in this chapter is widely adopted in multivariate time-series anomaly detection, where models are trained on normal

data and anomalies are rare or weakly labeled. In this setting, window-level scoring provides a natural interface for inference and for trading detection reactivity against computational cost, including feature processing and model latency (Zamanzadeh Darban et al., 2024b; Xu et al., 2022). Importantly for SDN, telemetry is often collected as periodically sampled sequences of counters and flow statistics, so the windowed representation aligns with how monitoring pipelines expose observations and how controllers and switches export measurements through standard APIs (Cevallos et al., 2024).

Building on this methodological perspective, the experimental progression of the thesis is intentionally two-stage. In Chapter 4, the intent was to adopt supervised and classical benchmarking baselines to quantify dataset difficulty, establish reproducible reference points and identify potentially feature dependencies based on generation-based biases under a narrowed set taxonomy of attacks. However, supervised baselines implicitly rely on an idealized threat model in which sufficient labeled data are available for all relevant classes. In practice, SDN operators rarely possess large and up-to-date labeled datasets, while emergent behaviours and zero-day attacks are the norm. This motivates the second stage of the thesis, which evaluates detection under limited labels and emphasizes generalization to previously unseen behaviours (Zamanzadeh Darban et al., 2024b; Ataa et al., 2024b). Therefore, this chapter will provide an additional analysis using anomaly detection in order to evaluate robustness under two specific conditions: limited labelling and zero-day attacks. In addition, this chapter will specifically apply the sliding window formulation to KRONOS-SDN and investigate deep reconstruction-based detection, while comparing four representative autoencoder architectures through a controlled comparison under homogeneous training, scoring and reporting conditions, so that any differences between architectures are due to architecture, and not to experimental variables.

Therefore, this chapter extends the analysis of KRONOS-SDN through a reconstruction-based anomaly detection setting designed to evaluate robustness under limited labelling and zero-day-oriented conditions. The sliding-window formulation is applied to per-entity traffic sequences ex-

tracted from KRONOS-SDN packet captures, and four representative autoencoder architectures are compared under homogeneous training, scoring and reporting conditions. This controlled setup ensures that observed differences are attributable primarily to architectural design rather than to inconsistent experimental variables.

## 6.2 Research questions and experimental focus

Using the information provided in the previous section, the anomaly detection study of KRONOS-SDN was developed guided by the following research questions:

- **RQ1 (Model expressiveness vs. efficiency).** How do the performances of different types of deep autoencoder architectures (convolutional CNN-AE, recurrent LSTM-AE, transformer-based Tr-AE and feedforward MLP) differ in terms of their ability to identify anomalies in SDN traffic windows, given that all models are trained on identical data, using the same representation of features? The four types of architectures are motivated in Section 5.3.4. The choice is characterized by increasing level of complexity and a transition from architectures that can be efficiently run on CPUs to those designed to be run on GPUs.
- **RQ2 (Real-time suitability).** Given a fixed hardware budget, what throughput (windows per second) and per-window latency can each model sustain when applied to KRONOS-SDN traffic and how does this interact with the cost of feature extraction from packet captures?
- **RQ3 (Resource footprint).** How do the architectures differ with respect to their use of CPU and RAM resources during detection and what trade-offs arise between the level of resource usage and the accuracy of the detection?
- **RQ4 (Architectural reuse).** Among the evaluated architectures, which model has the best balance between detection stability, op-

erational efficiency and flexibility to downstream multi-source fusion? Can such a model serve as a unified backbone for both traffic-based and host-telemetry-based anomaly signals within a cross-plane IDS framework?

To answer these questions, four families of autoencoder architectures were evaluated using the same sliding-window representation of traffic entities extracted from KRONOS-SDN packet captures and trained on benign entity-level windows without supervision. Comparability is ensured through a shared reconstruction-error scoring function and a threshold selected as a high quantile of the benign score distribution, so that all architectures are assessed under the same decision logic.

## 6.3 Sliding-window pipeline

The anomaly detection pipeline operates on features derived from packet captures collected in the KRONOS-SDN testbed. Packets are grouped into monitored traffic entities according to a fixed identifier. In this chapter, an entity is functionally defined by the source-destination IP pair, that is to say `(src_ip, dst_ip)`. This choice reflects a trade-off: while port-level granularity would increase the number of entities and the memory overhead during windowing, source-destination aggregation is sufficient to capture volume-based and temporal-pattern anomalies in SDN telemetry and aligns with the exploratory analysis in Chapter 4, where this representation proved effective for attack classification.

For each entity, a streaming frontend aggregates packets into fixed-length time ticks of duration $\Delta t$ seconds and computes a compact vector of statistics per tick. In the configuration used here, the vector comprises eight features: packets per second, bytes per second, mean and standard deviation of inter-arrival times, mean and standard deviation of packet sizes, mean activity time and mean idle time for the entity during the tick.

These features were selected based on the exploratory analysis of KRONOS-SDN reported in Chapter 4 and are intended to be interpretable,

computationally inexpensive to extract and representative of traffic characteristics relevant to intrusion detection. In particular, throughput indicators, such as packets and bytes per second, capture volume shifts; temporal and size statistics capture changes in traffic dynamics and activity/idle times expose behavioural shifts in entity persistence.

The resulting tick sequence is defined separately for each active entity at the tick resolution $\Delta t$ introduced in Subsection 5.4.1. Let $\mathcal{E}$ denote the set of monitored entities observed in the trace. For each entity $e \in \mathcal{E}$, the corresponding tick sequence is denoted as

$$\{\mathbf{x}_e(t)\}_{t=1}^{L_{\max}^{(e)}}$$

where $t$ is the discrete tick index for entity $e$, and $L_{\max}^{(e)}$ denotes the total number of ticks available for that entity. Each vector $\mathbf{x}_e(t) \in \mathbb{R}^d$ contains the feature values computed for entity $e$ at tick $t$, with $d = 8$ in the traffic-only experiments.

Given a window length $L$, the runtime sliding-window manager maintains an independent buffer for each active entity. As a result, windows are not constructed by mixing ticks belonging to different entities, but by preserving the temporal evolution of each entity separately. During the initial warm-up phase of an entity, corresponding to its first $L - 1$ ticks, the buffer is not yet full and no complete window is emitted for that entity.

Once the buffer contains $L$ ticks, a new runtime window is emitted whenever a new tick is available for that entity. Each emitted window is defined as

$$\mathbf{X}_{e,t} = [\mathbf{x}_e(t - L + 1), \ldots, \mathbf{x}_e(t)] \in \mathbb{R}^{L \times d}, \qquad t \geq L \tag{6.1}$$

Each window therefore represents the most recent $L$ consecutive ticks of a single monitored entity over $d$ feature channels. The parameter $L$ determines the amount of temporal context included in each sample, corresponding to an approximate real-time duration of $L \cdot \Delta t$. Instead, increasing the value associated to $\Delta t$ reduces the number of emitted ticks and model evaluations, whereas decreasing it provides finer-grained mon-

itoring at higher computational cost. Once the buffer is full, runtime windows advance by one emitted tick for each active entity.

Each window is standardized using scaling parameters estimated only from the training data. The implementation adopts an incremental standardization procedure that maintains running estimates of the mean and variance and applies the resulting transformation before model inference. All autoencoder architectures operate on standardized windows with the same $(L, d)$ shape, so that architectural comparisons are performed under a common input representation.

The choice of the tick resolution $\Delta t$ and the window length $L$ encodes a fundamental trade-off between detection delay, temporal context and computational cost. The tick resolution $\Delta t$ determines the temporal granularity at which traffic evolution is observed: smaller values provide finer-grained monitoring, but increase the number of ticks to process and may amplify short-term variability. The window length $L$ determines how much past information is included in each input sample. Short windows react more quickly and improve coverage of short-lived entities, whereas long windows provide richer temporal context at the cost of a longer warm-up phase, larger per-entity state and potentially higher detection latency.

In SDN environments, where near-real-time detection is needed to prevent attacks from overwhelming the controller or the data plane, the values of $\Delta t$ and $L$ should therefore be selected jointly, taking into account the required detection latency, the granularity of the available telemetry, the expected duration of the target attacks and the computational resources available at deployment time (Ali et al., 2023; Ferrag et al., 2020).

## 6.4 Autoencoder architectures

All four models considered in this study share a common high-level principle: given an input window $\mathbf{X}_\tau$, an encoder $f_\theta$ maps it to a latent representation $\mathbf{z}$ and a decoder $g_\phi$ attempts to reconstruct the original window, producing $\widehat{\mathbf{X}}_\tau = g_\phi(f_\theta(\mathbf{X}_\tau))$. The model parameters $(\theta, \phi)$ are trained to minimize a reconstruction loss, the mean absolute error between $\mathbf{X}_\tau$ and

$\widehat{\mathbf{X}}_\tau$. During detection, the reconstruction error on the most recent time step in the window provides an anomaly score for the corresponding entity and timestamp.

### 6.4.1 Multi-layer perceptron autoencoder (MLP-AE)

The MLP autoencoder constitutes the simplest architecture in our comparison. The MLP-AE exhibits several appealing properties for SDN deployment: it is straightforward to implement and calibrate, offers highly predictable computational cost per window, finally it is amenable to efficient execution on both CPU and modest GPU hardware. In this work, the same MLP-AE design is later reused to process windows of usage metrics (CPU, RAM, entropy, interface counters) from `collectd`, enabling a unified view of context-aware anomaly detection in KRONOS-SDN. Providing a more detailed description of this architecture thus facilitates the subsequent discussion of cross-plane models.

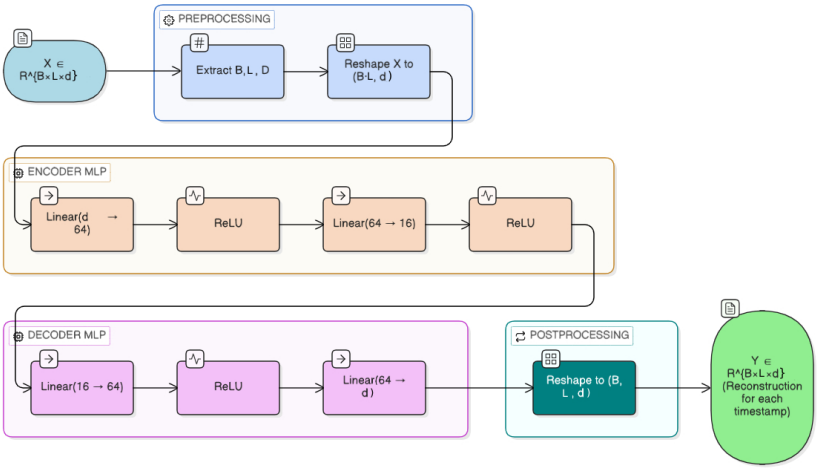



**Figure 13:** MLP architecture.

Figure 13 details the exact data flow of the adopted MLP model. Given an input tensor $\mathbf{X} \in \mathbb{R}^{B \times L \times d}$ (batch size $B$, window length $L$, and $d$ features per timestamp), a lightweight preprocessing step extracts $(B, L, d)$

and reshapes the input to $(B \cdot L,\ d)$ so that the same fully-connected encoder is applied independently to each timestamp (time-distributed MLP). The encoder consists of two linear projections $d \to 64$ and $64 \to 16$, each followed by a Rectified Linear Unit (ReLU) activation, producing a latent representation $\mathbf{Z} \in \mathbb{R}^{B \cdot L \times 16}$. The decoder mirrors this structure with $16 \to 64$ (ReLU) and $64 \to d$, yielding $\hat{\mathbf{X}} \in \mathbb{R}^{B \cdot L \times d}$, which is finally reshaped back to $\hat{\mathbf{X}} \in \mathbb{R}^{B \times L \times d}$ to provide a reconstruction for each timestep in the window.

### 6.4.2 Convolutional autoencoder (CNN-AE)

The convolutional autoencoder treats each window as a multichannel one-dimensional signal of length $L$ with $d$ channels. The encoder consists of a stack of temporal convolutional layers with progressively increasing receptive field, each followed by non-linear activation and (optionally) temporal pooling. These layers gradually reduce the temporal resolution and expand the channel dimension, yielding a high-level representation that captures local temporal patterns such as bursts, periodicities, or short anomalies.

The bottleneck representation is flattened and passed through a small fully connected block to obtain a latent vector $\mathbf{z} \in \mathbb{R}^h$, where $h$ denotes the latent dimensionality. The decoder mirrors this structure by using transposed convolutions and upsampling layers to reconstruct the original $L \times d$ window. Owing to the locality of convolutional filters, the CNN-AE balances expressiveness and computational efficiency, particularly for medium-length windows. Figure 14 depicts the CNN-AE instance adopted. Each $L \times d$ window is first mapped through an *input projection* (a linear layer) to a higher-dimensional channel space, then transposed to the channel-first layout required by 1-Dimensional Convolution layer (`Conv1d`), that, concretely, instead of reconstructing each timestep independently, reconstructs each point in the sequence using information from adjacent timesteps within a sliding temporal window, allowing the model to capture local trends, bursts and transient behaviours in the data. The core is a temporal *convolutional tower* made of four identical

blocks; each block applies two successive `Conv1d` layers, each followed by `ReLU` and dropout, so that stacking blocks increases the effective receptive field while preserving the temporal length. Finally, the activations are transposed back and an *output projection* maps the channels back to $d$ to obtain the reconstructed window.

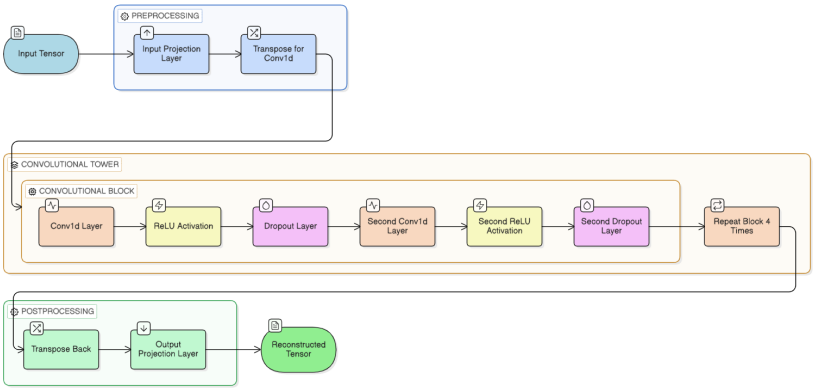



**Figure 14:** CNN architecture.

### 6.4.3 LSTM autoencoder (LSTM-AE)

The LSTM autoencoder represents a more classical recurrent approach. The encoder is a uni- or bi-directional LSTM that processes the window sequentially and compresses it into the final hidden state, which serves as the latent vector $\mathbf{z}$. The decoder is another LSTM initialized with $\mathbf{z}$ that attempts to regenerate the original sequence step by step.

Recurrent autoencoders have the advantage of directly modeling temporal order and can be effective on relatively short windows. However, their sequential nature limits the degree of parallelism and can result in higher training and inference times compared to MLP-based model.

Figure 15 summarizes the LSTM-AE variant adopted in this thesis. Each sliding-window is a sequence $\mathbf{X} \in \mathbb{R}^{N\times 8}$ (with $N$ time steps and $8$ features per step) that is first mapped through an *input projection* followed by ReLU. The resulting embedded sequence is then processed by

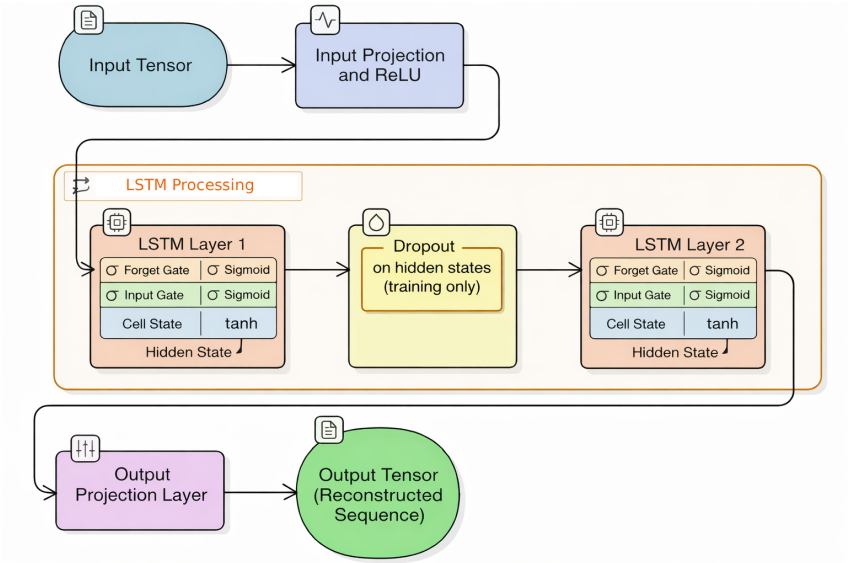



**Figure 15:** LSTM architecture.

two stacked LSTM layers; a dropout stage is applied *only during training* to the inter-layer hidden-state outputs to improve generalization. The top LSTM produces a hidden-state sequence whose final state provides the compact summary $\mathbf{z}$, while an *output projection* maps the LSTM outputs back to the original feature dimension to obtain the reconstructed window $\hat{\mathbf{X}} \in \mathbb{R}^{N \times 8}$.

### 6.4.4 Transformer autoencoder (Tr-AE)

The transformer autoencoder operates on the same $L \times d$ windows but replaces convolutions with multi-head self-attention layers. First, each time step $\mathbf{x}(t)$ is projected into a $d_{\text{model}}$-dimensional embedding and augmented with a positional encoding that allows the model to distinguish different positions within the window. A stack of transformer encoder blocks applies self-attention and position-wise feed-forward layers to these embeddings, enabling the model to capture long-range temporal

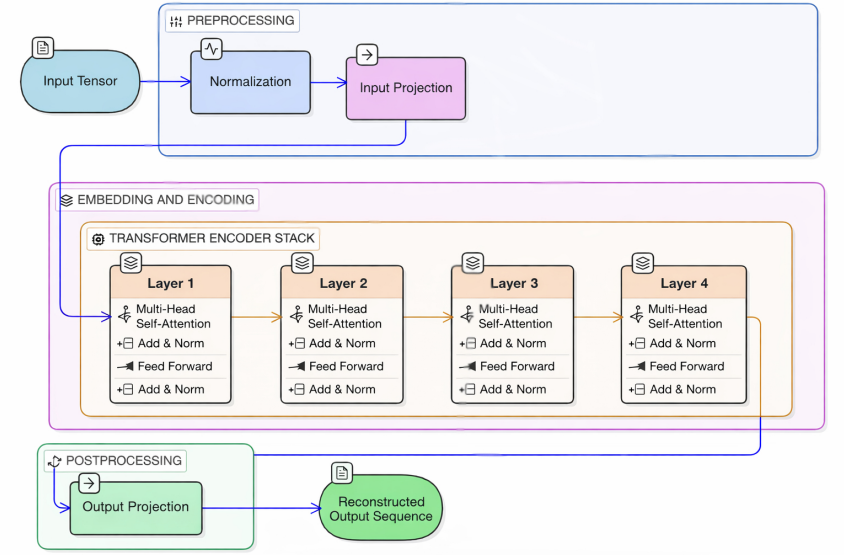



**Figure 16:** Transformer Architecture.

dependencies and complex feature interactions.

To obtain a fixed-size latent representation, the final encoder embeddings are aggregated via average pooling over time or select the embedding of a dedicated "latent" token prepended to the sequence. The decoder then reconstructs the sequence using either a symmetric transformer decoder or a lighter stack of feed-forward layers applied to the encoder outputs. Thanks to its parallelism, the Tr-AE can exploit GPUs efficiently and support larger window sizes, at the cost of increased memory usage.

Figure 16 reports the Tr-AE instance adopted. Each input window $\mathbf{X} \in \mathbb{R}^{L \times d}$ is first *normalized* and then mapped through an *input projection* $\mathbb{R}^{d} \to \mathbb{R}^{d_{\text{model}}}$, yielding a sequence of $d_{\text{model}}$-dimensional embeddings. The core model is a stack of $4$ Transformer blocks, each composed of multi-head self-attention followed by a position-wise feed-forward sublayer, both wrapped by residual connections and layer normalization (*Add & Norm*). The resulting contextualized embeddings are finally mapped

back to the original feature space through an *output projection* $\mathbb{R}^{d_{\text{model}}} \to \mathbb{R}^{d}$ to produce the reconstructed sequence $\hat{\mathbf{X}} \in \mathbb{R}^{L \times d}$.

## 6.5 Static threshold baseline

All models are trained in an *unsupervised manner using windows extracted from benign traffic segments* in KRONOS-SDN. The pipeline consists of two stages: *offline pretraining and threshold selection on benign data*, and *online detection with a fixed threshold*.

**Offline pretraining and fixed-threshold selection.** Given a benign training set $\mathcal{D}_{\text{train}} = \{\mathbf{X}_\tau\}$, model parameters are learned by minimizing an L1 reconstruction loss:

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{X}_\tau \in \mathcal{D}_{\text{train}}} \frac{1}{(L-1)d} \sum_{t=2}^{L} \sum_{j=1}^{d} |x_j(t) - \widehat{x}_j(t)| \tag{6.2}$$

using mini-batch stochastic optimization. After training, the model is frozen and anomaly scores $s(t)$ (Eq. 6.3) are computed on benign windows to obtain an empirical score distribution. A static decision threshold is then selected via a quantile rule:

$$\tau_{\text{static}} = q_\alpha\big(s(t)\big)$$

with $\alpha = 0.995$ for the fixed-threshold baseline.

**Online detection (single phase, fixed rule).** During detection, for each incoming window ending at time $t$, the score $s(t)$ is computed and the window is flagged as anomalous if $s(t) > \tau_{\text{static}}$. Alerts are logged together with the entity identifier, timestamp and the per-window feature-processing.

## 6.6 Window-level scoring and labelling policy

**Anomaly score.** Given a window $\mathbf{X}_{e,t}$ ending at tick $t$, the autoencoder outputs a reconstruction $\widehat{\mathbf{X}}_{e,t}$. A scalar anomaly score is defined consis-

tently across all architectures. In order to align scoring with near-real-time decision semantics, the reconstruction error on the most recent tick in the window is adopted. Specifically, the score is computed as the mean absolute reconstruction error over the $d$ feature channels of the last tick:

$$s(e,t) \triangleq \frac{1}{d}\sum_{j=1}^{d} |x_{e,j}(t) - \widehat{x}_{e,j}(t)| \tag{6.3}$$

where $\mathbf{x}_e(t)$ and $\widehat{\mathbf{x}}_e(t)$ denote the last row of $\mathbf{X}_{e,t}$ and $\widehat{\mathbf{X}}_{e,t}$, respectively. This last-tick scoring rule assigns the anomaly score to the same timestamp at which the runtime decision is emitted.

**Window labelling.** Performance metrics are computed at the window level. Let $\mathcal{I} = \{[t_{\text{start}}^{(j)}, t_{\text{end}}^{(j)}]\}$ be the set of ground-truth attack intervals. A window ending at time $t$ is labeled as malicious if its end timestamp falls within any attack interval:

$$y(t) = \begin{cases} 1 & \exists j : \; t_{\text{start}}^{(j)} \leq t \leq t_{\text{end}}^{(j)} \\ 0 & \text{otherwise.} \end{cases} \tag{6.4}$$

This deterministic rule avoids ambiguity at interval boundaries and matches the decision time associated with the score in Eq. 6.3.

## 6.7 Performance and efficiency metrics

The current section describes the group of metrics that will be utilized in evaluating the quality of detection as well as the operation of an IDS, using the same metrics described in Section 5.2, and these will be calculated at the *window level* and where applicable at the *attack phase* in order to allow for comparisons across different phases of a campaign.

### 6.7.1 Detection performance metrics

Detection quality is reported through precision, recall, $F_1$ score and the area under the ROC curve (AUC-ROC). In addition to global summaries,

class-conditional metrics for the attack class ($y = 1$, according to the notation employed in the Equation 6.4) are reported ($P_1$, $R_1$, $F1_1$), since operational IDS decisions are primarily determined by the trade-off between false alarms and missed attacks (Howe et al., 2025; L. Yang et al., 2024).

### 6.7.2 Efficiency and resource metrics

Operational viability is characterized by the following indicators.

**Feature extraction cost.** For each window, the wall-clock time required to convert raw packets into the entity-level representation by the detector is measured. This includes packet parsing, aggregation into fixed-duration ticks and computation of the traffic statistics used to build the normalized $L \times d$ window representation.

**Model latency and throughput.** Inference efficiency is quantified via: per-window inference latency, consisting of the forward pipeline producing $\widehat{\mathbf{X}}_\tau$ and the scalar anomaly score; throughput, expressed as processed windows per second. Throughput is reported both as *model-only*, isolating the inference stage, and as *end-to-end wall-clock* throughput when the full pipeline is considered.

**Resource usage.** During detection runs, CPU utilization and RAM occupancy are monitored and summarized over the run. GPU is used only for training.

### 6.7.3 Measurement protocol and logging

Timing metrics are collected using a consistent measurement protocol. Feature-extraction time per window is measured as the wall-clock duration required to parse packets, update tick statistics and emit a normalized $L \times d$ window. Inference time per window is measured as the wall-clock duration of the forward pass and score computation. CPU/RAM are sampled periodically with fixed cadence and aggregated as mean and

standard deviation over the run. All metrics are logged to per-dump CSV files to enable fine-grained post-hoc analysis and cross-model comparison.

## 6.8 Benchmark evaluation on the dataset and reference model selection

This section reports a controlled benchmark on the dataset in order to assess detection quality and deployability across deep architectures and motivate the selection of a single reference model for the subsequent context-modulation study. To prevent unequal degrees of freedom across architectures, the same training protocol is applied to all models and only two interpretable factors are varied in a compact $2 \times 2$ design: window duration ($w \in \{60, 120\}$ s) and training schedule (40 epochs with a learning rate equal to $5 \cdot 10^{-4}$ vs 60 epochs at $3 \cdot 10^{-4}$), with batch size fixed to 128. In this context, an epoch denotes one complete pass of the training algorithm over the available training windows, while the batch size indicates the number of windows processed before each parameter update.

### 6.8.1 Benchmark design and run comparability

The benchmark is organized into four controlled experimental settings, identified by the IDs A–D and used consistently throughout the remainder of this section. Each setting is defined by the window length $w$, the number of training epochs, the learning rate and the batch size. In Table 18, precisely in the compact run identifier, `w` denotes the window length, the value following `e` denotes the number of epochs, `lr` denotes the learning rate and `bs` denotes the batch size. The term *support size* refers to the number of samples available in each run, where each sample corresponds to one generated sliding window. Accordingly, $N$ denotes the total number of windows, while $N_0$ and $N_1$ denote the number of normal and attack windows, respectively. Reporting these quantities makes it possible to verify that the benchmark settings are comparable in terms of data support and class composition.

**Table 18:** Run identifiers and dataset support sizes.

| ID | Run | $w$ (s) | epochs | lr | bs | $N$ | $N_0$ | $N_1$ | attack % |
|---|---|---|---|---|---|---|---|---|---|
| A | gpu_w60e40lr5e-4bs128 | 60 | 40 | $5 \cdot 10^{-4}$ | 128 | 2 242 586 | 2 001 325 | 241 261 | 10.76% |
| B | gpu_w60e60lr3e-4bs128 | 60 | 60 | $3 \cdot 10^{-4}$ | 128 | 2 242 586 | 2 001 325 | 241 261 | 10.76% |
| C | gpu_w120e40lr5e-4bs128 | 120 | 40 | $5 \cdot 10^{-4}$ | 128 | 2 221 558 | 1 982 069 | 239 489 | 10.78% |
| D | gpu_w120e60lr3e-4bs128 | 120 | 60 | $3 \cdot 10^{-4}$ | 128 | 2 221 558 | 1 982 069 | 239 489 | 10.78% |

Table 18 shows that the support sizes are million-scale and nearly identical across runs. For $w = 60$, the total number of generated windows is $N = 2{,}242{,}586$, with $N_1 = 241{,}261$ attack windows (10.76%). For $w = 120$, the total support is $N = 2{,}221{,}558$, with $N_1 = 239{,}489$ attack windows (10.78%). The resulting stability in class prevalence ($\approx 10.76$–$10.78\%$) supports interpreting differences in subsequent tables as effects of the controlled factors and architectural choices, rather than as artifacts of varying class priors or insufficient support.

### 6.8.2 Training schedule: rationale for epochs and learning rate

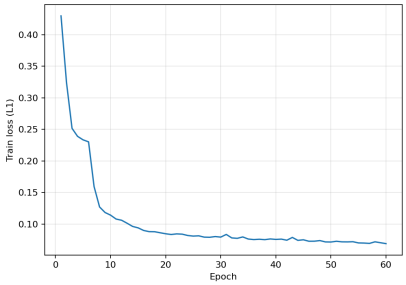
The selection of the number of epochs and of the learning rate was driven by two constraints: ensuring fair and comparable training conditions across heterogeneous autoencoder families and limiting hyper-parameter degrees of freedom to avoid implicit tuning for each architecture. Allowing per-model tuning would introduce an additional and, at the same time, difficult to quantify confounder, since heterogeneous autoencoder families exhibit different sensitivities to hyper-parameters and effectively imply different search-space sizes, potentially turning the comparison into a proxy for how extensively each model was tuned rather than how it generalizes under a fixed training budget. In addition, adopting a fixed training protocol improves repeatability, since all architectures are evaluated under the same optimization conditions. Moreover, since the anomaly-detection setting does not provide a reliable supervised validation signal for model selection, the schedule was not tuned to maximize a label-dependent metric; rather, it was guided by empirical convergence behaviour and by the need to enforce a single, architecture-agnostic pro-

tocol across all experiments.

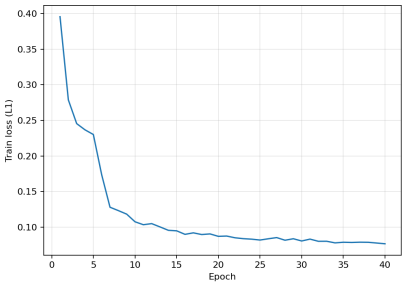
**Learning-rate range and stability.** The learning rate (LR) is a primary factor in optimization: excessively small values slow down convergence, while large values may lead to unstable loss dynamics and even to divergence. This sensitivity is well established in the deep-learning literature and motivates keeping the learning-rate search space narrow and conservative when the objective is controlled model comparison rather than hyper-parameter optimization (Wu et al., 2023). In addition, LR tuning is recognized as a major practical cost in deep-learning-based models development; a recent work explicitly targets reducing this tuning burden by automatically adapting the LR scale factor (Cutkosky et al., 2023). Accordingly, the study was restricted to the two close and stable LR values reported in Table 18, avoiding architecture-specific tuning.

**Epoch budget, worst-case criterion and diminishing returns.** Rather than selecting the epoch budget independently for each model, it was derived from a conservative worst-case criterion. In particular, the CNN was taken as the reference architecture for schedule selection because it exhibited the slowest and most sensitive convergence behaviour, especially when increasing the window duration from $w = 60\,\mathrm{s}$ to $w = 120\,\mathrm{s}$. The CNN training-loss curves (Figures 17-18) show a rapid decrease during the first $\sim$10-15 epochs followed by a long regime of diminishing returns, with a clear saturation trend after approximately 25-35 epochs. Based on this empirical saturation pattern, the two epoch budgets were set to 40 and 60 epochs to guarantee convergence even in the most demanding setting while keeping the computational cost bounded; extending training beyond these values yields only marginal improvements in reconstruction loss.

**Coupled schedules to balance speed and refinement.** To isolate schedule effects with minimal degrees of freedom, the epoch budget is deliberately coupled to the LR: a shorter schedule with a slightly larger step size

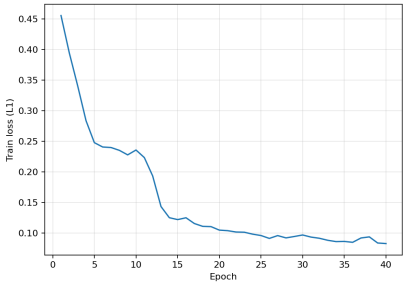



**(a)** Training loss for CNN-AE training with W=60 ticks, epochs = 40 and lr = $5 \cdot 10^{-4}$.

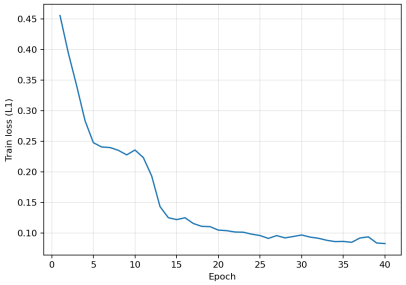



**(b)** Training loss for CNN-AE training with W=60 ticks, epochs = 60 and lr = $3 \cdot 10^{-4}$.

**Figure 17:** Training loss for CNN-AE training with W=60 ticks of 1s.




**(a)** Training loss for CNN-AE training with W=120 ticks, epochs = 40 and lr = $5 \cdot 10^{-4}$.




**(b)** Training loss for CNN-AE training with W=120 ticks, epochs = 60 and lr = $3 \cdot 10^{-4}$.

**Figure 18:** Training loss for CNN-AE training with W=120 ticks of 1s.

and a longer schedule with a slightly smaller step size, as reported in Table 18, reflecting standard optimization practice (Wu et al., 2023). Overall, the adopted grid is intentionally small and fixed *a priori* to preserve experimental comparability and to ensure that subsequent architectural conclusions are not artifacts of extensive hyper-parameter search.

### 6.8.3 Detection performance across architectures

Window-level detection performance is summarized through global metrics (accuracy, macro-$F_1$, AUC) and attack-class metrics ($P_1$, $R_1$, $F1_1$).

**Table 19:** Classification reports for each run and model. $P_1$, $R_1$, $F1_1$ refer to the attack class ($y = 1$).

| Model (-AE) | ID | Acc | Macro-F1 | AUC | $P_1$ | $R_1$ | $F1_1$ | Support$_1$ |
|---|---|---|---|---|---|---|---|---|
| **MLP** | A | **0.9660** | **0.9219** | 0.9895 | 0.7614 | 0.9964 | **0.8632** | 241 261 |
| | B | 0.9589 | 0.9073 | 0.9887 | 0.7270 | 0.9892 | 0.8381 | 241 261 |
| | C | 0.9617 | 0.9129 | 0.9893 | 0.7416 | 0.9891 | 0.8476 | 239 489 |
| | D | **0.9662** | **0.9225** | 0.9897 | **0.7625** | 0.9973 | **0.8643** | 239 489 |
| **CNN** | A | 0.9283 | 0.7978 | 0.9827 | 0.7009 | 0.5812 | 0.6355 | 241 261 |
| | B | **0.9636** | **0.9141** | 0.9874 | **0.7666** | 0.9510 | **0.8489** | 241 261 |
| | C | 0.9574 | 0.9052 | 0.9879 | 0.7166 | 0.9998 | 0.8349 | 239 489 |
| | D | 0.9603 | 0.9052 | 0.9862 | 0.7621 | 0.9184 | 0.8330 | 239 489 |
| **LSTM** | A | 0.9496 | 0.8905 | **0.9903** | 0.6809 | **1.0000** | 0.8102 | 241 261 |
| | B | 0.9373 | 0.8689 | **0.9903** | 0.6317 | **1.0000** | 0.7743 | 241 261 |
| | C | **0.9674** | **0.9251** | **0.9903** | **0.7680** | **1.0000** | **0.8688** | 239 489 |
| | D | 0.9643 | 0.9188 | **0.9903** | 0.7514 | **1.0000** | 0.8580 | 239 489 |
| **Transformer** | A | 0.9602 | 0.9041 | 0.9876 | **0.7655** | 0.9082 | 0.8307 | 241 261 |
| | B | 0.9357 | 0.8269 | 0.9822 | 0.7174 | 0.6641 | 0.6897 | 241 261 |
| | C | 0.9541 | 0.8856 | 0.9831 | 0.7607 | 0.8373 | 0.7972 | 239 489 |
| | D | 0.9475 | 0.8643 | 0.9865 | 0.7539 | 0.7622 | 0.7580 | 239 489 |

Table 19 together with Figures 19 and 20 indicate that MLP-AE exhibits the most stable performances across settings: macro-$F_1$ remains within $[0.9073, 0.9225]$ and AUC remains in a narrow range (0.9887–0.9897). Within the responsive $w = 60$ regime, run A, MLP-AE improves the attack-class $F1_1$ with respect to run B (0.8632 vs 0.8381), driven by higher precision (0.7614 vs 0.7270) while maintaining very high recall (0.9964 vs 0.9892). LSTM achieves $R_1 = 1.0$ in all runs and reaches the highest macro-$F_1$ in run C (0.9251), but at the cost of substantially elevated false-alarm rates visible in the confusion analysis conducted in Subsection 6.8.4. Transformer performance is more sensitive to the setting (macro-$F_1$ drops to 0.8269 in run B), reducing suitability as a single stable baseline for downstream analyses.

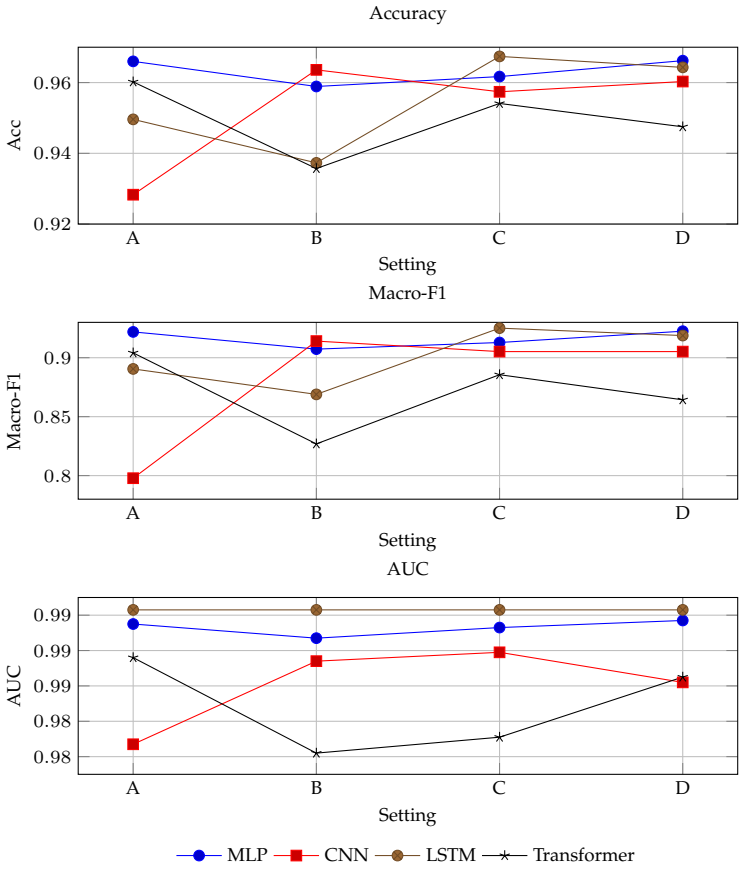



**Figure 19:** Overall metrics across settings A–D.

### 6.8.4 Error structure and operational trade-offs

Confusion counts and derived error rates (FPR and FNR) are reported to expose operationally relevant trade-offs, separating alert volume (false positives) from missed attacks (false negatives).

Table 20 and the graphical representation in Figure 21 show that MLP-AE provides a bounded error profile across all settings: FNR remains between 0.27% and 1.09% while FPR remains between 3.75% and 4.48%.

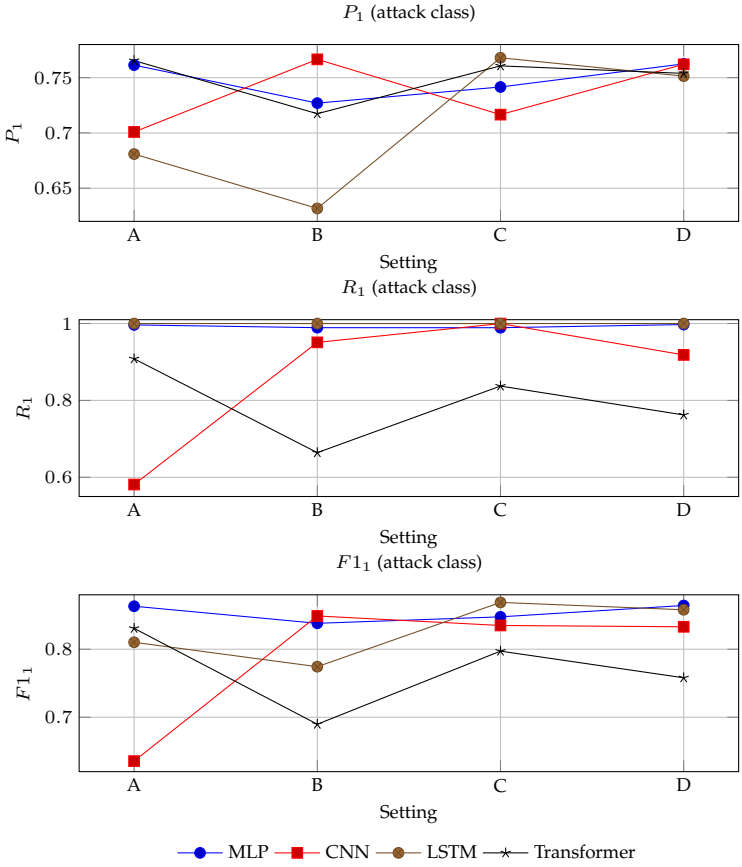



**Figure 20:** Attack-class metrics across settings A–D.

Within $w = 60$ s, run A improves over run B on both axes simultaneously (FPR 3.76% vs 4.48%; FNR 0.36% vs 1.08%), supporting the adoption of run A as the primary configuration. LSTM drives FNR to $\approx 0\%$ in all runs, but this can correspond to increased false alarms (FPR 7.03% in run B). CNN and Transformer exhibit stronger regime dependence (e.g., CNN run A FNR 41.88%; Transformer run B FNR 33.59%).

**Table 20:** Confusion metrics for each run and model, FPR=FP/(TN+FP), FNR=FN/(FN+TP).

| Model (-AE) | ID | TN | FP | FN | TP | FPR (%) | FNR (%) |
|---|---|---|---|---|---|---|---|
| **MLP** | A | 1 925 991 | 75 334 | 879 | 240 382 | 3.76 | 0.36 |
| | B | 1 911 731 | 89 594 | 2 612 | 238 649 | 4.48 | 1.08 |
| | C | 1 899 526 | 82 543 | 2 608 | 236 881 | 4.16 | 1.09 |
| | D | 1 907 689 | 74 380 | 638 | 238 851 | 3.75 | 0.27 |
| **CNN** | A | **1 941 489** | **59 836** | 101 044 | 140 217 | **2.99** | 41.88 |
| | B | 1 931 487 | **69 838** | 11 815 | 229 446 | 3.49 | 4.90 |
| | C | 1 887 386 | 94 683 | 36 | 239 453 | 4.78 | 0.02 |
| | D | 1 913 404 | 68 665 | 19 550 | 219 939 | 3.46 | 8.16 |
| **LSTM** | A | 1 888 258 | 113 067 | **1** | **241 260** | 5.65 | **0.00** |
| | B | 1 860 678 | 140 647 | **0** | **241 261** | 7.03 | **0.00** |
| | C | 1 909 739 | 72 330 | **5** | **239 484** | 3.65 | **0.00** |
| | D | 1 902 819 | 79 250 | **0** | **239 489** | 4.00 | **0.00** |
| **Transformer** | A | 1 934 183 | 67 142 | 22 144 | 219 117 | 3.35 | 9.18 |
| | B | **1 938 199** | 63 126 | 81 031 | 160 230 | **3.15** | 33.59 |
| | C | **1 918 984** | **63 085** | 38 965 | 200 524 | **3.18** | 16.27 |
| | D | **1 922 476** | **59 593** | 56 942 | 182 547 | **3.01** | 23.78 |

### 6.8.5 Latency and resource footprint

Per-window latency is decomposed into feature extraction and model inference and resource usage is summarized through CPU and RAM sampling.

Table 21 shows that feature extraction is negligible and stable across architectures (Feat ms $\approx 0.013$–$0.022$), indicating that deep inference dominates the per-window cost. MLP-AE provides the lowest inference latency across settings ($\approx$ 1–1.5 ms), whereas CNN/LSTM/Transformer incur substantially higher costs (roughly 9–41 ms depending on the model and setting). CPU and RAM footprints are broadly comparable across models.

### 6.8.6 Throughput and end-to-end timing

Throughput is reported both as model-only throughput (isolating inference) and as end-to-end wall-clock throughput, together with average

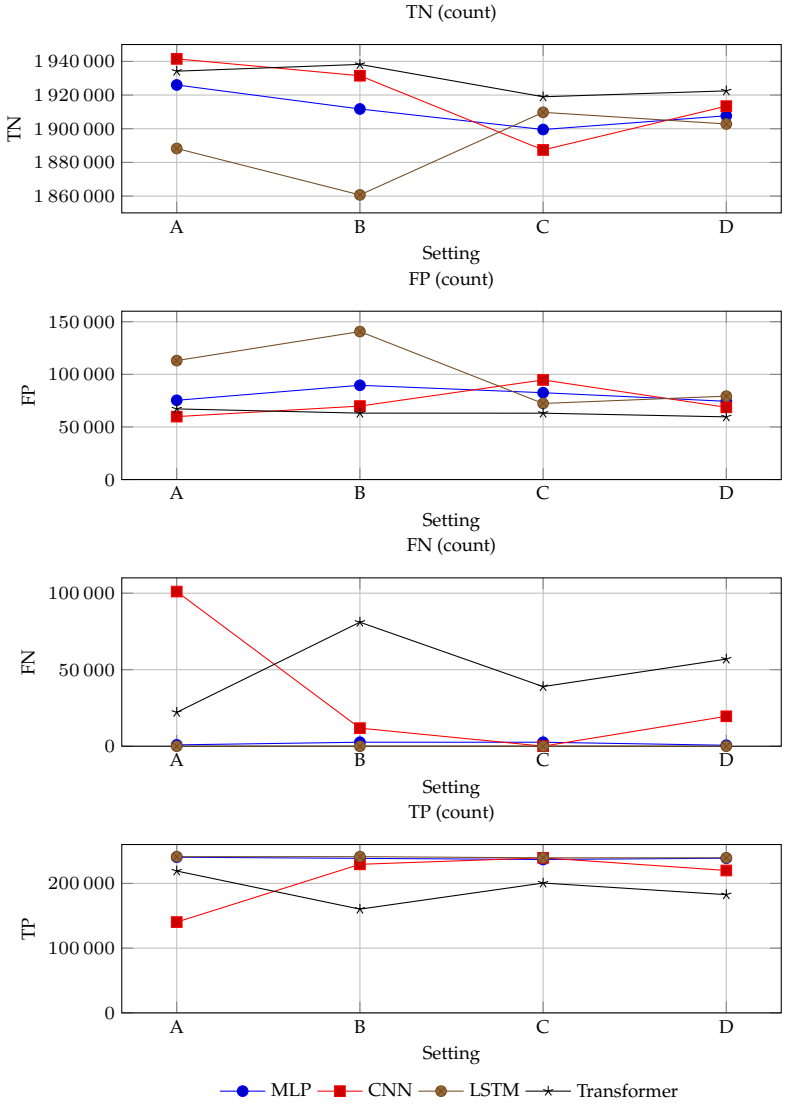



**Figure 21:** Confusion-matrix counts across settings A–D (TP, FP, FN, TN).

**Table 21:** Latency and resource footprint (mean±std) for each run and model.

| Model (-AE) | ID | Infer ms | Feat ms | CPU (%) | RAM (MB) |
|---|---|---|---|---|---|
| **MLP** | A | **1.26±2.03** | 0.0195±0.3047 | 75.26±7.65 | 495.1±12.7 |
| | B | **1.01±1.27** | 0.0148±0.1818 | 79.80±6.16 | 500.6±15.5 |
| | C | **1.48±2.23** | 0.0194±0.3001 | 75.89±9.54 | 433.6±101.3 |
| | D | **1.53±2.28** | 0.0191±0.3000 | 74.58±7.85 | 492.6±15.0 |
| **CNN** | A | 16.82±7.91 | 0.0184±0.2845 | **57.71±8.34** | 517.1±22.6 |
| | B | 12.72±5.13 | 0.0144±0.1756 | **65.69±4.08** | 521.7±31.4 |
| | C | 27.75±12.70 | 0.0195±0.3028 | **57.65±8.99** | 385.9±124.0 |
| | D | 27.99±12.82 | 0.0193±0.2993 | **58.27±9.23** | 513.1±51.0 |
| **LSTM** | A | 11.42±5.97 | **0.0170±0.2548** | 58.63±9.18 | **477.9±45.9** |
| | B | 8.97±4.21 | **0.0135±0.1647** | 68.37±6.00 | **483.7±84.6** |
| | C | 20.01±9.60 | **0.0146±0.2054** | 60.53±9.07 | **336.3±164.1** |
| | D | 20.28±9.77 | **0.0148±0.2056** | 60.77±9.39 | **394.3±114.6** |
| **Transformer** | A | 21.75±10.20 | 0.0220±0.3429 | 59.74±9.47 | 505.2±12.0 |
| | B | 16.32±6.50 | 0.0164±0.2031 | 67.21±6.98 | 510.9±14.7 |
| | C | 40.85±18.59 | 0.0202±0.3124 | 60.59±11.86 | 466.9±86.7 |
| | D | 40.97±18.24 | 0.0200±0.3085 | 59.87±9.83 | 503.5±16.6 |

total time per scored window.

**Table 22:** Throughput and end-to-end timing across the four experimental settings.

| Model (-AE) | Metric | A | B | C | D |
|---|---|---|---|---|---|
| **MLP** | Win/s (model-only) | **1809.2±325.7 (0.18)** | **2188.3±191.3 (0.09)** | **1722.4±434.4 (0.25)** | **1631.8±295.3 (0.18)** |
| | Win/s (end-to-end, wall) | **114.6±103.1 (0.90)** | **152.7±135.2 (0.89)** | **110.5±102.4 (0.93)** | **109.6±99.9 (0.91)** |
| | Avg total time (ms, scored) | **4.12±0.92** | **2.99±0.33** | **4.29±1.26** | **4.47±1.01** |
| **CNN** | Win/s (model-only) | 67.4±16.6 (0.25) | 85.0±5.9 (0.07) | 39.9±9.8 (0.24) | 39.4±9.1 (0.23) |
| | Win/s (end-to-end, wall) | 31.1±25.9 (0.83) | 38.1±29.8 (0.78) | 20.7±16.6 (0.80) | 20.4±16.2 (0.79) |
| | Avg total time (ms, scored) | 19.59±4.60 | 14.67±0.90 | 30.57±7.26 | 30.79±7.00 |
| **LSTM** | Win/s (model-only) | 99.7±23.0 (0.23) | 126.3±31.6 (0.25) | 57.4±15.7 (0.27) | 56.5±15.3 (0.27) |
| | Win/s (end-to-end, wall) | 39.5±33.2 (0.84) | 47.6±37.1 (0.78) | 26.2±20.9 (0.80) | 25.8±20.5 (0.79) |
| | Avg total time (ms, scored) | 13.84±3.65 | 10.57±2.03 | 21.89±6.57 | 22.14±6.46 |
| **Transformer** | Win/s (model-only) | 50.7±10.0 (0.20) | 65.8±1.5 (0.02) | 26.6±6.2 (0.23) | 26.5±6.0 (0.23) |
| | Win/s (end-to-end, wall) | 24.4±19.4 (0.80) | 30.9±23.8 (0.77) | 14.9±11.3 (0.76) | 14.8±11.1 (0.75) |
| | Avg total time (ms, scored) | 25.01±4.72 | 18.53±0.43 | 44.29±9.71 | 44.41±9.48 |

Table 22 reports the numerical throughput and end-to-end timing results across the four experimental settings. Values are reported as mean±std; for Win/s, the coefficient of variation, computed as $\mathrm{CV} = \mathrm{std}/\mathrm{mean}$, is reported in parentheses to quantify runtime vari-

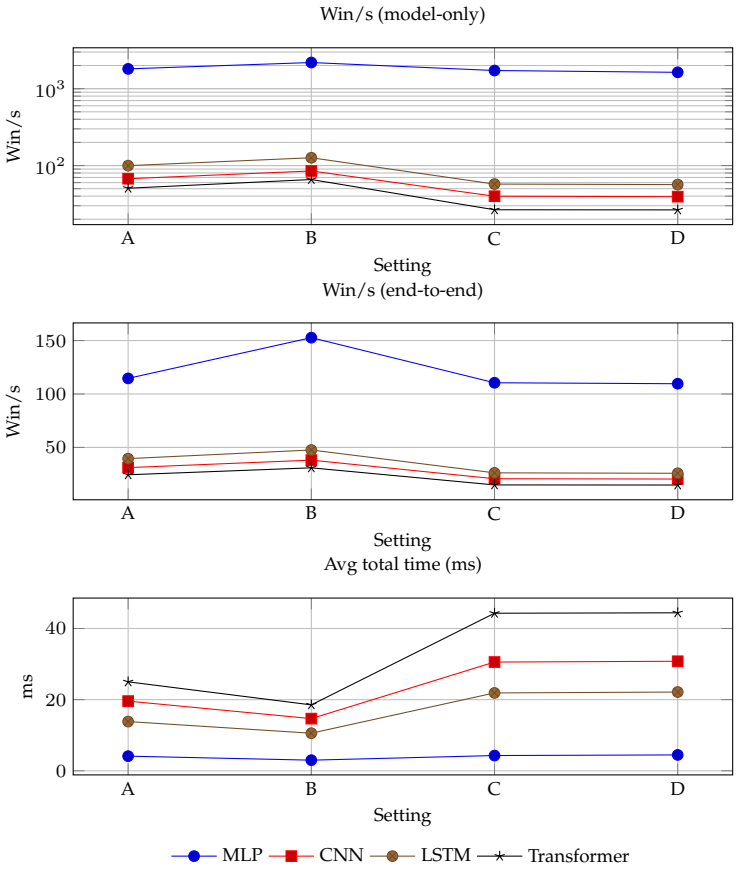



**Figure 22:** Throughput and timing across settings A–D.

ability. Figure 22 provides a graphical comparison of the corresponding average trends across models and settings.

The results confirm that MLP-AE provides the strongest efficiency profile under both model-only and end-to-end measurements: model-only throughput is in the thousands of windows/s (e.g., 2188.3 in run B) and average total time per scored window remains in the 3–4.5 ms range (2.99 ms in B; 4.12 ms in A). CNN and Transformer require substantially larger per-window budgets, with Transformer reaching approximately

44 ms in runs C and D. The systematic drop from model-only to wall-clock throughput across all architectures, together with the high end-to-end variability reported in Table 22 (CV typically $\approx 0.75$–$0.93$), indicates that wall-clock throughput is dominated by system-level effects. Under such conditions, selecting a baseline detector with minimal inference cost is the most conservative choice for downstream analyses.

### 6.8.7 Reference model and configuration for context modulation

The combined evidence across Tables 19–22 motivates fixing a single reference detector and configuration for the subsequent context-modulation study. MLP-AE is selected as the reference architecture because it jointly provides:

- stable and high detection quality across the full benchmark grid (Table 19);
- a bounded and operationally safe confusion profile with low missed-attack rates and moderate false-alarm rates (Table 20)
- the lowest inference latency and best throughput profile among the evaluated architectures (Tables 21 and 22).

In summary, MLP-AE provides the elements required in the RQ4 (Architectural reuse) provided at Section 6.2.

Setting A ($w = 60$ s, 40 epochs, $5 \cdot 10^{-4}$, bs=128) is adopted as the primary configuration for the next phase. This choice prioritizes responsiveness through the shorter window length and is supported by the best MLP-AE error trade-off within the $w = 60$ regime (lower FPR and lower FNR than setting B in Table 20) while maintaining a very low operational cost.

The results of the comparative study provide a baseline understanding of how different deep autoencoder architectures behave when applied solely to entity-based traffic windows extracted from KRONOS-SDN. In the following section, the same sliding-window machinery and

the MLP-AE design are leveraged, due to its superior performances jointly with reduced latency time, to incorporate host-level metrics from `collectd`, building a cross-plane anomaly detection framework that integrates data-plane and control/host-plane signals within a unified detection pipeline.

### 6.8.8 Context-modulated dynamic thresholding

As anticipated in Section 4.9, a key limitation of reconstruction-based anomaly detectors is that their decision boundary is typically implemented through a *single*, fixed threshold $\tau_0$ on the distribution of the reconstruction error computed in train phase. In operational SDN deployments, however, both the traffic observed in the data plane and the workload experienced by the controller in the control plane are non-stationary: legitimate bursts, maintenance activities and benign workload fluctuations may increase the reconstruction error even in the absence of attacks, while low-rate or stealthy adversarial actions may remain below a conservative threshold. Recent work in anomaly detection and cyber-physical monitoring has highlighted that static thresholds can undermine reliability and lead to unacceptable false-alarm rates, motivating *context-aware* and *adaptive* thresholding strategies (Aoudi et al., 2021b; X. Yang et al., 2023).

In this thesis, this idea is realized through a **cross-plane coupling** mechanism that augments a data-plane reconstruction-based detector with host-level telemetry from the SDN controller virtual machine. Specifically, let $e(t)$ denote the *anomaly score* at time $t$, computed as the reconstruction error of the sliding window ending at $t$ (i.e., larger $e(t)$ indicates that the observed traffic window is less consistent with the benign patterns learned during training). Let $c(t)$ be the *context vector* collecting controller-side host metrics at time $t$. Rather than changing the score, the context is used to adapt the decision boundary: the effective threshold becomes

$$\tau(t) = \tau_0 + g\big(c(t)\big)$$

where $g(\cdot)$ modulates the baseline threshold $\tau_0$ according to the current

operating regime. A window is flagged as anomalous whenever $e(t) > \tau(t)$.

This design *does not replace* the traffic-based detector, but *modulates* its sensitivity at runtime. Here, $\text{Entropy}^{\text{ctrl}}(t)$ denotes the host-level entropy metric collected on the SDN controller host at time $t$. In particular, it refers to the available kernel entropy reported by the operating system and monitored through `collectd`. When KRONOS-SDN telemetry reports a benign but unusual controller-side state, for example when $\text{Entropy}^{\text{ctrl}}(t)$ exceeds a given quantile while the traffic mix remains consistent with normal behaviour, $g(c(t))$ raises the threshold to prevent spurious alarms due to telemetry-induced variability in reconstruction errors. Conversely, when controller and host metrics indicate an unusually stressed or inconsistent state, $g(c(t))$ lowers the threshold to increase sensitivity, thereby favouring early detection of subtle attacks that manifest as small deviations in reconstruction error under adversarial conditions.

This cross-plane coupling mechanism is aligned with emerging strategies in SDN security, where collaborative evidence from different planes improves robustness (Han et al., 2018; Yue et al., 2022).

#### Host-telemetry context model

**Telemetry collection.** Controller-side resource and activity indicators are acquired through `collectd` (collectd Project, 2025), including CPU, memory and system activity metrics. While early prototypes explored additional engineered features (e.g., discrete derivatives and peak counters), in the final architecture such operations are intentionally excluded because they were not used in the reported experiments and increase the noise of the context score without consistent gains. Instead, a **feature-selection-first** strategy is adopted, which improves interpretability and stability.

**Feature selection and rationale.** Empirically, using a wide set of resource counters produced an overly noisy context score, strongly reacting only to high-impact attacks (e.g., intense brute force or DoS/DDoS),

while being less informative for milder scenarios. Therefore the input space is reduced to two telemetry signals that exhibited a stable correlation with attack windows targeting the ONOS controller across the week:

- the kernel entropy level (`entropy-entropy`);
- the interrupt activity on a network-associated IRQ line (in our setup, `irq-17`).

Intuitively, entropy provides a compact indicator of structured system activity (including cryptographic and I/O-related events), whereas IRQ activity captures a more direct proxy of network-driven kernel work, which is closely related to control-plane stress under anomalous traffic.

**Windowing, scaling and autoencoder.** Let $x(t) \in \mathbb{R}^D$ be the telemetry vector at second $t$ (after feature selection). Sliding windows of length $W_{\text{ctx}}$ seconds with hop $H_{\text{ctx}}$ is built and each window is flattened into a vector in $\mathbb{R}^{W_{\text{ctx}}D}$. A global standardization is computed on the training windows and reused at inference time. Then a compact MLP autoencoder (Hinton et al., 2006) is trained to minimize a reconstruction loss (MAE). For each context window a scalar reconstruction error is computed

$$s_{\text{ctx}}(t) = \text{MAE}(\mathbf{x}_{\text{ctx}}(t), \widehat{\mathbf{x}}_{\text{ctx}}(t)) \tag{6.5}$$

which is interpreted as a continuous **context severity**:

$$ctx\_severity(t) \triangleq s_{\text{ctx}}(t) \tag{6.6}$$

This is consistent with common reconstruction-based anomaly scoring practices (Sakurada et al., 2014; Chalapathy et al., 2019).

**Discretizing context severity into levels via multi-threshold Otsu**

To couple the continuous severity signal with a multi-threshold traffic detector, $ctx\_severity(t)$ is discretized into $L$ ordered levels:

$$ctx\_level(t) \in \{0, 1, \ldots, L-1\}$$

Rather than selecting thresholds manually, **multi-level Otsu thresholding** is applied on the empirical distribution of $s_{\text{ctx}}$ computed over training windows. Otsu's method selects thresholds that maximize between-class variance (equivalently, minimize within-class variance) in a histogram-based partition (Otsu, 1979). For multi-level thresholding the standard extension is adopted and efficient formulations used in the literature (Liao et al., 2001). Denoting the resulting $L-1$ thresholds as $\{\theta_1, \ldots, \theta_{L-1}\}$, the mapping is:

$$ctx\_level(t) = \begin{cases} 0 & s_{\text{ctx}}(t) \leq \theta_1 \\ 1 & \theta_1 < s_{\text{ctx}}(t) \leq \theta_2 \\ \vdots & \\ L-1 & s_{\text{ctx}}(t) > \theta_{L-1} \end{cases}$$

All $\theta_k$ are stored in the context model checkpoint to ensure deterministic inference.

**Daily context dynamics under attack**

Figure 23 reports the daily evolution of the continuous context severity signal ($ctx\_severity$, blue) together with its discretized operating-regime level ($ctx\_level$, orange, piecewise-constant), while the shaded regions mark the ground-truth attack intervals. Across the week, attacks targeting the controller consistently induce pronounced departures from the benign regime: $ctx\_severity$ exhibits sharp spikes or sustained plateaus that push $ctx\_level$ toward the highest bins, confirming that the selected host telemetry captures controller-side stress conditions that occur with malicious activity.

On Day 1, the attack interval was characterized by an increase in the level of $ctx\_severity$ and, consequently, a sustained increase in $ctx\_level$. Furthermore, it is important to notice that after the end of the shaded area representing the actual attack period, $ctx\_severity$ did not rapidly return to its pre-attack base line but instead slowly decreased over time. Therefore, this demonstrates a *recovery latency* where the controller remained in a stressed state even though the active phase of the attack had ended.

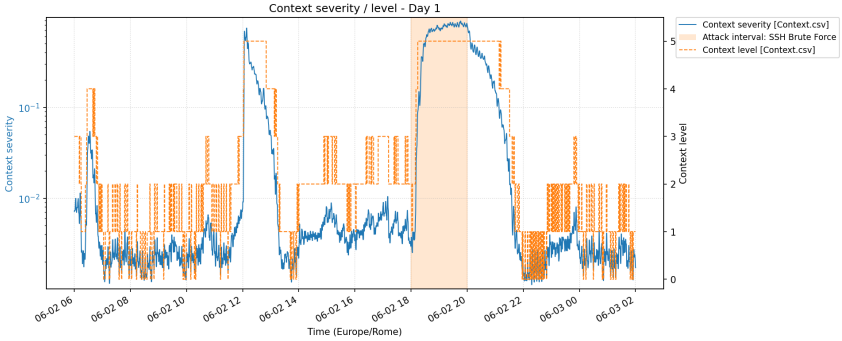



**Figure 23:** Context severity and level for Day 1.

It is important to highlight that in Figure 23, two transient peaks around 06:00 and 12:00 fall outside the shaded ground-truth attack interval and are therefore not labeled as controller-targeting attacks. These excursions are instead consistent with *host discovery* and *service discovery* activities directed toward data-plane components (i.e., reconnaissance of hosts and exposed services), rather than direct adversarial actions against the controller itself. Interestingly, despite not being aimed at the controller, such data-plane probing can still propagate to the controller-side telemetry by inducing additional control-plane workload (e.g., reactive handling and monitoring triggered by atypical traffic patterns), which in turn raises $ctx\_severity$ and temporarily increases $ctx\_level$. This observation suggests that, at least in principle, the proposed context signal could also help capture anomalous traffic phenomena that primarily involve the data plane and do not explicitly target the controller, reinforcing the value of cross-plane observability.

For completeness, Appendix D provides the plots for context severity and discretized level for each experimental day.

### Traffic anomaly detector with multi-quantile thresholds

The second component is a PCAP-based detector operating on entity-level features extracted from network traces. An MLP autoencoder is

*trained on benign traffic windows* (same training interval policy as the context model) and compute a reconstruction-based anomaly score $s_{\text{pcap}}(t)$. Unlike fixed-threshold baselines, **multiple thresholds** are stored as quantiles of the benign score distribution:

$$\tau_{\text{levels}} = \big[\tau_0, \tau_1, \ldots, \tau_{L-1}\big], \quad \tau_\ell = q_{\alpha_\ell}\big(s_{\text{pcap}}\big)$$

where $\alpha_\ell$ are increasing quantile levels (e.g., in the experiments a fine-grained set is persisted from $q_{99.0}$ to $q_{99.9}$). This "multi-quantile" representation enables runtime sensitivity control without retraining.

Here, each $\tau_\ell$ is a *scalar* threshold expressed in the same units as the anomaly score $s_{\text{pcap}}(t)$. Thus, $\tau_{\text{levels}}$ is a discrete set of admissible operating points: at runtime, the detector does not continuously "learn" a new threshold, but it selects one of the pre-computed quantile cutoffs $\{\tau_0, \ldots, \tau_{L-1}\}$, each corresponding to a target tail probability $\alpha_\ell$ on the benign score distribution.

**Runtime cross-plane coupling and dynamic threshold selection**

At detection time, for each traffic window ending at time $t$:

- $s_{\text{pcap}}(t)$ is computed;
- the aligned $ctx\_level(t)$ is retrieved from telemetry (using the same temporal grid and the window end timestamp).

Finally, the **dynamic threshold** $\tau(t)$ is defined as the element of $\tau_{\text{levels}}$ indexed by the current context severity level:

$$\tau(t) = \tau_{\text{levels}}\big[ctx\_level(t)\big] \tag{6.7}$$

Equation (6.7) makes explicit that $\tau(t)$ can only take values in the finite set $\{\tau_0, \ldots, \tau_{L-1}\}$. In other words, $\tau(t)$ is a piecewise-constant function over time: it changes only when $ctx\_level(t)$ changes. With the adopted ordering, $ctx\_level(t) = 0$ denotes the most conservative operating point (which in cascade implies the largest threshold on the

flow network model), whereas larger $ctx\_level(t)$ correspond to progressively more sensitive operating points (lower thresholds) selected under increasing controller-side suspiciousness.

Formally, this is enforced by storing $\tau_{\text{levels}}$ in monotone order such that $\tau_0 \geq \tau_1 \geq \cdots \geq \tau_{L-1}$ (equivalently, $\alpha_0 \geq \alpha_1 \geq \cdots \geq \alpha_{L-1}$).

The final decision becomes:

$$anomaly_{\text{pcap}}(t) = \begin{cases} 1 & s_{\text{pcap}}(t) > \tau(t) \\ 0 & \text{otherwise} \end{cases}$$

In summary, the aforementioned mechanism implements a practical form of context-aware thresholding, conceptually related to adaptive thresholding frameworks proposed in adjacent domains (Aoudi et al., 2021b; X. Yang et al., 2023).

**Output format.** For each analyzed traffic window the following data is recorded: timestamp, entity, score_pcap, tau_dynamic, anomaly_pcap, ctx_severity, ctx_level.
This makes the coupling auditable and supports post-hoc interpretation (e.g., whether an alert was enabled by the context state).

**Experimental evidence: with vs. without context**

The effect of context modulation is evaluated by comparing a **without-context** baseline using a single fixed threshold, namely $q_{99.5}$, with a **with-context** configuration in which $\tau(t)$ varies between conservative and aggressive quantiles, from $q_{99.9}$ down to $q_{99.0}$, as a linear function of $ctx\_level(t)$, while keeping the same training regime and test set.

Table 23 reports the class-recall comparison between the two configurations. Normal traffic, corresponding to class 0, is aggregated in a single row across all files, whereas anomalous traffic, corresponding to class 1, is reported separately for each attack time slot. In the table, $R$ denotes the class recall computed from the confusion matrix, while the relative percentage change is computed as $\Delta_R(\%) = (R^{ctx} - R^{no\text{-}ctx})/R^{no\text{-}ctx} \cdot 100$.

Tables 24 and 25 summarize the resulting window-level classification performance. Context modulation yields a consistent improvement on

**Table 23:** Recall comparison between fixed-threshold and context-modulated detection.

| Traffic | Support | $R^{\text{no-ctx}}$ (%) | $R^{\text{ctx}}$ (%) | $\Delta_R$ (%) |
|---|---|---|---|---|
| Normal (0) | 2026420 | 96.73 | 96.82 | **+0.09** |
| SSH Brute Force | 6578 | 81.68 | 89.40 | **+9.45** |
| HTTP Brute Force | 18448 | 83.62 | 89.76 | **+7.34** |
| LDAP Amplification | 27669 | 100.00 | 100.00 | +0.00 |
| DDoS (SYN Flood) - ONOS | 38906 | 99.99 | 100.00 | +0.01 |
| Flow table flooding | 38976 | 99.99 | 100.00 | +0.01 |
| DoS | 11830 | 100.00 | 100.00 | +0.00 |
| DDoS (UDP Flood) | 23902 | 100.00 | 100.00 | +0.00 |
| DDoS (SYN Flood) - Switches | 38160 | 100.00 | 100.00 | +0.00 |
| HTTP DoS + DDoS (SYN Flood) | 15513 | 100.00 | 100.00 | +0.00 |
| DDoS (SYN Flood) - Switch + ONOS | 13673 | 100.00 | 100.00 | +0.00 |
| DDoS | 28643 | 100.00 | 100.00 | +0.00 |

the attack class (label 1), increasing precision and recall simultaneously and it reduces both false positives and false negatives. In particular, the number of false positives decreases from $16{,}415$ to $11{,}753$ (approximately $-28.4\%$), while false negatives decrease from $4{,}226$ to $2{,}798$ (approximately $-33.8\%$), indicating that the adaptive thresholding mechanism improves sensitivity without sacrificing operational reliability.

### Discussion

This section investigated a deployment-oriented form of cross-plane coupling in which controller-side host telemetry is used to modulate the sensitivity of a traffic-only reconstruction-based detector. It is important to remind that the approach does not introduce a second detection stage: the PCAP autoencoder remains the sole source of the anomaly score, while context affects only the decision boundary through a time-varying threshold $\tau(t)$. Operationally, a context model produces an operating-regime indicator that is mapped to a set of pre-computed quantile thresholds learned from benign traffic and the detector flags an anomaly when-

**Table 24:** Classification report comparison between context-modulated thresholding and a fixed-threshold baseline.

| Scenario | Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|---|
| With context | 0 | 0.9968 | 0.9869 | 0.9918 | 896386 |
| | 1 | 0.8094 | 0.9469 | 0.8727 | 52695 |
| | Macro avg | 0.9031 | 0.9669 | 0.9323 | 949081 |
| | Weighted avg | 0.9864 | 0.9847 | 0.9852 | 949081 |
| | Accuracy | – | – | 0.9847 | 949081 |
| Without context | 0 | 0.9952 | 0.9817 | 0.9884 | 896386 |
| | 1 | 0.7470 | 0.9198 | 0.8244 | 52695 |
| | Macro avg | 0.8711 | 0.9507 | 0.9064 | 949081 |
| | Weighted avg | 0.9814 | 0.9783 | 0.9793 | 949081 |
| | Accuracy | – | – | 0.9783 | 949081 |

**Table 25:** Confusion matrix comparison.

| Scenario | True class | Pred 0 | Pred 1 |
|---|---|---|---|
| With context | 0 | 884633 | 11753 |
| | 1 | 2798 | 49897 |
| Without context | 0 | 879971 | 16415 |
| | 1 | 4226 | 48469 |

ever the traffic score exceeds the context-selected threshold.

The results on KRONOS-SDN show that this lightweight modulation improves robustness under non-stationary conditions. Compared to a fixed-threshold baseline, the context-aware configuration reduces false positives while also decreasing false negatives for those attacks that target the application plane or are characterized by stealthy network behaviour, yielding a better overall error trade-off without retraining the traffic model. These gains support the core hypothesis that cross-plane observability can stabilize reconstruction-based decisions in realistic SDN environments, where legitimate workload fluctuations may otherwise trigger spurious alarms and conservative static thresholds may hide low-rate attacks.

In conclusion, the context-adaptive thresholding is a viable and effec-

tive fusion method; it maintains the ease of use and comparable metrics of a pure traffic-scoring system, introduces a very small amount of additional computational complexity (regime lookup and selecting a threshold), and provides a very clear way to determine how to modify the anomaly detection based on the current operational state of the SDN controller.

# Chapter 7

# Conclusions and further work

Software-Defined Networking (SDN) alters the design space of intrusion detection by coupling security monitoring with network-wide observability and programmatic interaction with active network elements. The availability of controller-driven state and telemetry (e.g., flows, topology, and control events) enables tighter integration between detection and policy enforcement.

However, SDN also intrinsically introduces some vulnerabilities that must be taken into consideration; the control plane and its communications are high value targets and overloading or compromising them could potentially cause a network-wide issue or in extreme cases, disruption of the network. Therefore, intrusion detection should not solely be viewed as an offline classification problem: rather, it should be seen as a deployable function where the quality of detection should be evaluated in conjunction with the time it takes for alerts to be generated, the throughput of the system and the overall compute resource utilization required by the system.

The thesis evolved from identifying gaps in typical experimental methods used to evaluate SDN security, then creating a benchmark substrate to enable evaluation and finally developing and testing near-real-time

detection methodologies within the limits of realistic deployment constraints.

## 7.1 Summary of contributions

The thesis provides three main contributions.

**Evidence-driven analysis of SDN IDS evaluation practices.** The thesis first revisits the assumptions and limitations that recur in widely used SDN intrusion detection datasets and experimental protocols, showing why many reported results are difficult to generalize across topologies, workloads and threat conditions. The analysis highlights that the fidelity of the underlying SDN architecture to real deployments, the breadth and representativeness of the collected traffic and the diversity of attack classes and generation characteristics, together with labelling assumptions and the availability (or, in most cases, absence) of multi-plane observability, fundamentally determine which detection mechanisms can be meaningfully validated.

**Interpretability as a driver for dataset benchmarking.** Across the contributions above, a consistent theme emerges: in SDN, detection effectiveness cannot be separated from operational feasibility. The same architecture and scoring rule may be acceptable or unacceptable depending on latency budgets, resource contention with control-plane services and the stability of alerting under workload changes. For this reason, the thesis treats interpretability as a methodological safeguard rather than an optional add-on. Feature attribution is used to verify that model decisions are supported by distributed, semantically meaningful cues, rather than by fragile shortcuts or spurious correlations, to identify redundancy and to relate dataset-level properties to model behaviour in anomaly detection.

**KRONOS-SDN as a realistic and reproducible SDN benchmark substrate.** Building upon the above analysis, the thesis introduces KRONOS-

SDN as a benchmark dataset designed to support temporally coherent, multi-day evaluation, while enabling controlled experimentation and repeatability. KRONOS-SDN is structured to facilitate systematic studies of flow-based detection alongside controller and host telemetry, bridging the gap between network-level traces and operational signals that can explain or modulate detection behaviour.

**Deployment-aware deep anomaly detection with cross-plane stabilization.** The thesis concretizes sliding-window deep anomaly detection on KRONOS-SDN through a unified pipeline and a controlled comparison of representative autoencoder families (CNN-AE, LSTM-AE, Transformer-AE and MLP-AE) under homogeneous conditions. The evaluation explicitly takes also into consideration the costs that determine feasibility in realistic SDN deployments, including feature extraction overhead, inference latency, throughput stability and CPU/RAM utilization. In addition, the thesis introduces a lightweight context-modulation mechanism that uses controller-side telemetry to infer operating regimes and to select thresholds from benign quantiles, yielding a time-varying decision threshold $\tau(t)$ with minimal overhead. The resulting evidence indicates that cross-plane signals can improve robustness to workload variability, reduce spurious alerts under legitimate fluctuations and improve sensitivity to specific attack patterns without retraining the traffic model.

## 7.2 Limitations

Despite the effort to increase realism and to explicitly account for deployment constraints, several limitations remain.

First, while KRONOS-SDN expands beyond common single-burst and DDoS-centric patterns, no benchmark can exhaustively cover the SDN threat landscape. In particular, more extensive representations of multi-stage campaigns, SDN-specific manipulations (e.g., policy logic abuse) and stealthy lateral movement would further stress generalization.

Second, the cross-plane strategy explored in this thesis prioritizes lightweight coupling and deployability. As a result, it does not fully explore more expressive fusion architectures, nor does it yet cover the full range of signals available in practical deployments (e.g., richer switch-side indicators or controller event streams).

## 7.3 Future work: hierarchical detection and broader fusion

The most immediate extension of the results illustrated in the thesis is to deepen cross-plane and cross-domain integration while preserving near-real-time feasibility. A particularly promising direction, for which ongoing work is already producing encouraging evidence, is a hierarchical hybrid detection strategy that combines supervised and anomaly-based reasoning in a staged pipeline.

Specifically, the hierarchical approach in ongoing work adopts a two-stage design in which a supervised model filters known attack patterns with high precision and an autoencoder trained on benign traffic is then applied only to samples classified as normal by the first stage. This structure targets shortcomings derived both from supervised and reconstruction-based detectors: on the one hand, it can reduce the cost of running anomaly detection indiscriminately over all traffic, while explicitly addressing false negatives, on the other hand it can improve coverage under previously unseen attacks (e.g., via leave-one-out evaluation protocols). The empirical results obtained suggest that such hierarchies can substantially improve recall on unseen attacks while maintaining a controlled false-positive rate on benign traffic.

From the perspective of this thesis, hierarchical hybridization aligns naturally with context-modulated calibration. A concrete next step is therefore to integrate staged supervised/anomaly decision-making for known/unknown coverage and cross-plane context signals to stabilize thresholds and mitigate workload-driven false alarms. This combination would preserve the lightweight operational footprint of the modulation

mechanism while extending detection capability across a broader spectrum of threat novelty.

Beyond hierarchical hybridization, several additional research directions follow directly from the findings of this work:

- **Richer multi-plane observability and fusion:** extend telemetry beyond host-level controller metrics toward controller event streams and switch-side signals and investigate fusion mechanisms that remain robust to missing or delayed signals.
- **Drift-aware calibration under streaming constraints:** move from fixed benign quantiles toward adaptive calibration policies that remain stable under workload shifts and evolving traffic patterns, without frequent retraining.
- **Operational explainability at scale:** develop explanation summaries that are stable across time windows and entities and are aligned with analyst triage workflows.

## 7.4 Closing remarks

Overall, this thesis argues and empirically supports, that progress in ML-based SDN intrusion detection depends on realistic benchmarks, reproducible protocols and deployment-oriented objectives. By providing KRONOS-SDN as a realistic and reusable benchmark dataset, and by showing how deep anomaly detection can be evaluated jointly with efficiency metrics and stabilized under realistic constraints through lightweight cross-plane modulation, the thesis provides a concrete foundation for repeatable SDN IDS research, data and an evaluation workflow aligned with near-real-time requirements and a clear direction for future work: integrating learning-based detection with SDN-specific context signals to improve robustness and reliability as operating conditions evolve, while preserving strong accuracy, efficiency and interpretability.

# Appendix A

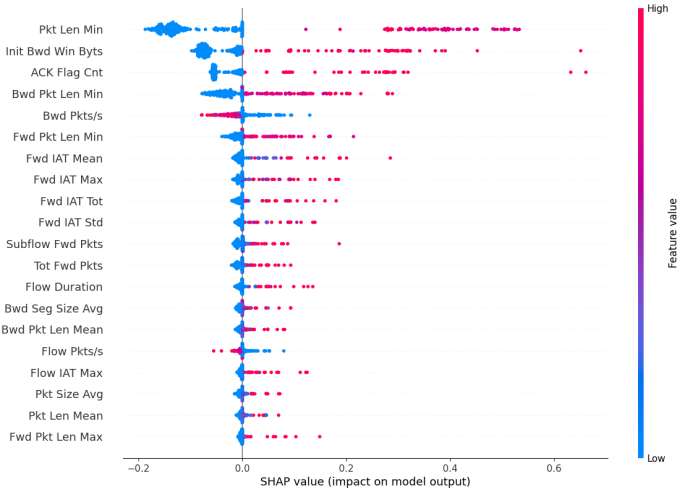
# InSDN DDoS SHAP Analysis

This appendix reports the post-hoc interpretability analysis conducted on the trained machine-learning classifiers using SHapley Additive exPlanations (SHAP). While the main body of the thesis focuses on predictive performance, SHAP analysis was adopted to complement these results by quantifying how each input feature contributes to the model decision, thereby enabling a more transparent comparison of the learned decision mechanisms across models.

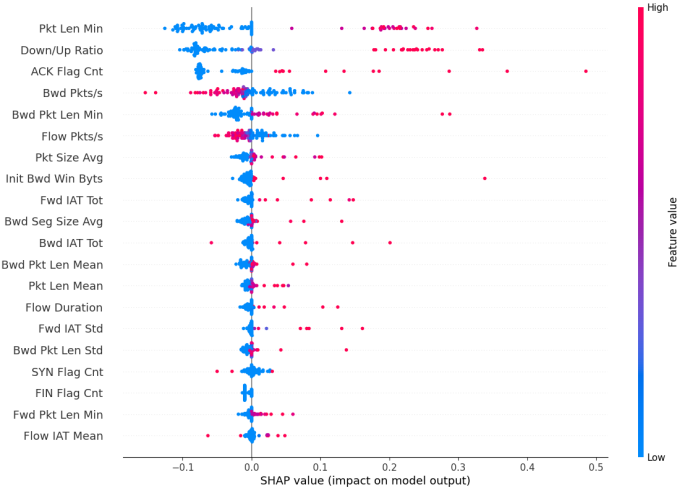
Figures A.1, A.2 and A.3 summarize the model-specific explanations through SHAP *beeswarm* plots. Each dot represents one sample; the horizontal position corresponds to the SHAP value (i.e., the signed contribution of that feature to the prediction), whereas color encodes the original feature value, from low to high. Features are sorted vertically by their mean absolute SHAP value, so the top rows denote the most influential variables globally. Positive SHAP values indicate that the feature increases the model output toward the anomalous/malicious class, in this case DDoS attack, while negative values push predictions toward the benign class. This visualization highlights both:

- the overall importance ranking;

- the directionality and variability of feature effects, offering insight into which traffic descriptors are consistently leveraged by different classifiers and where model behaviors diverge, as detailed in Subsection 3.2.5.

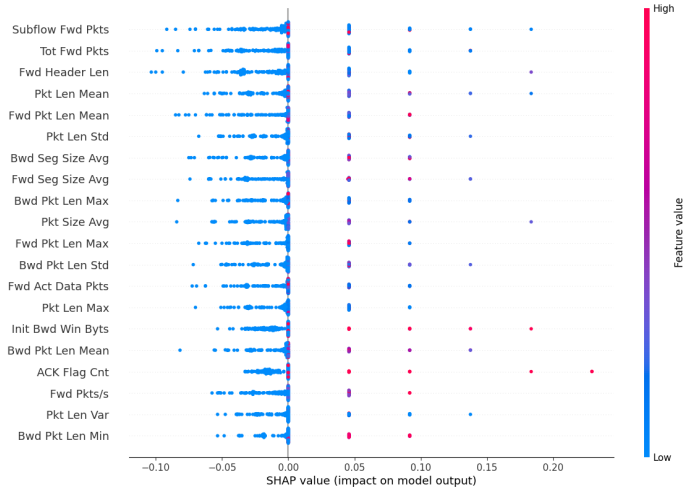



(a) SVM

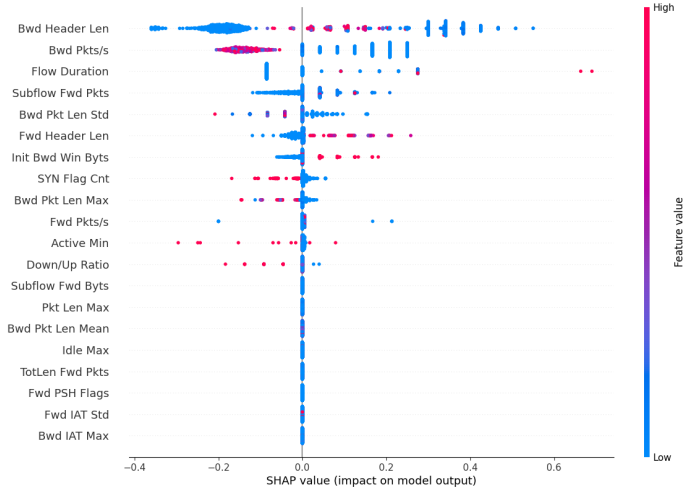



(b) k-NN

**Figure A.1:** Beeswarm SHAP plots for DDoS attack in InSDN dataset for SVM and k-NN.

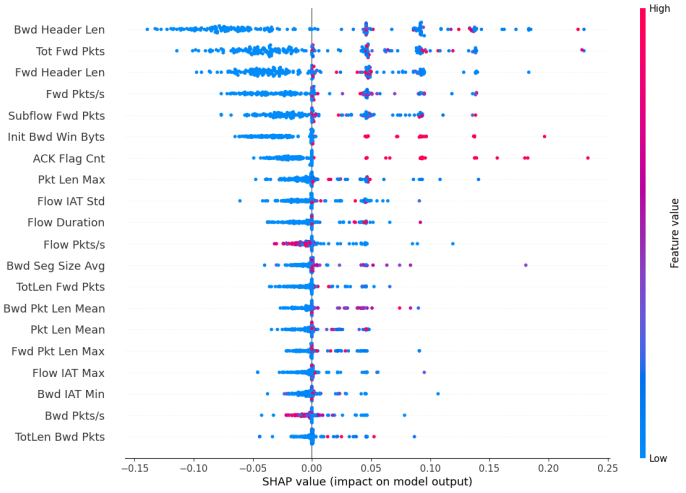



(c) Naive Bayes

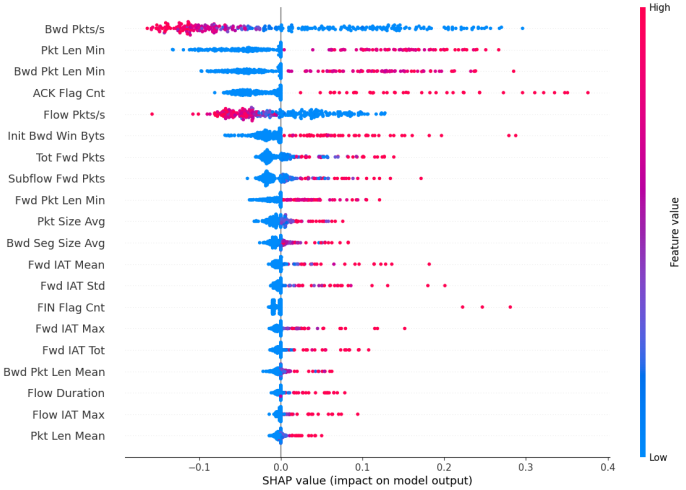



(d) Decision Tree

**Figure A.2:** Beeswarm SHAP plots for DDoS attack in InSDN dataset for NB and DT.




(e) Random Forest




(f) MLP

**Figure A.3:** Beeswarm SHAP plots for DDoS attack in InSDN dataset for RF and MLP.

# Appendix B

# Technical Details for the Malicious Traffic Generation

This appendix presents further technical specifics about the malicious traffic generation suite employed in KRONOS-SDN. Its purpose is to enrich the concise description provided in the main text by detailing, for each attack scenario, the generator adopted, the intended semantics of the attack, the destination endpoint, the packet-generation primitive, the nominal transmission pattern and the primary considerations affecting reproducibility.

The testbed uses an ONOS controller to operate Open vSwitch instances via the OpenFlow southbound interface. In the experimental setup, communication between the controller and the switches was made accessible through the OpenFlow control channel running on TCP port 6653. The attack-generation framework comprised multiple scenarios aimed at this segment of the infrastructure, such as *Packet Crafting OpenFlow*, *OpenFlow Traffic Injection*, *Flow Table Flooding* and *OpenFlow DoS: Flooding Flow Rules*. These scenarios were intended to stress the SDN control plane by sending crafted or high-rate traffic to the OpenFlow endpoint.

For reproducibility, it is helpful to clarify the protocol layer at which this traffic was generated. The Scapy-driven, OpenFlow-focused scenarios operated at the TCP stack level and emitted packets, optionally carrying arbitrary payloads, targeted at TCP port 6653. Their role was therefore to stress the OpenFlow-facing communication endpoint and its associated control-plane processing path, rather than to function as a complete OpenFlow application-layer message generator. In this setup, packets were not assembled using an OpenFlow protocol library, nor did they explicitly encode particular OpenFlow message types such as `PACKET_IN` or `FLOW_MOD`. Even so, this approach enables the dataset to capture how the SDN infrastructure behaves under malformed, highly bursty or flooding traffic aimed at the control channel, while clearly delineating the protocol boundary of the generated traffic.

Table B.1 summarizes the complete malicious traffic generation suite. The original dataset labels are preserved to maintain consistency with the released traces, labels and experimental schedule. For each scenario, the table lists the generator employed, the intended semantics, the endpoint, the packet primitive, the nominal sending pattern and a brief note on reproducibility. For burst-oriented attacks that are repeatedly triggered by the scheduler, the actual packet volume is influenced by the duration of each burst, the scheduler's relaunch interval, the load on the attacker's virtual machine and the hypervisor's scheduling behavior. Consequently, the reported rates should be understood as idealized packet-generation patterns. In the scenarios denoted as *Flow Table Flooding* and *OpenFlow DoS: Flooding Flow Rules*, these values represent the attempted load directed at the OpenFlow endpoint, rather than a directly observed count of installed flow rules.

**Table B.1:** Technical reproducibility summary of the malicious traffic generation suite.

| Attack label | Tool | Target semantics | Endpoint | Primitive and send pattern |
|---|---|---|---|---|
| Host Discovery | Nmap | Reconnaissance | Subnet-wide | ICMP/IP probes; one-shot sweep |
| OS Fingerprinting | Nmap | Reconnaissance | ONOS controller | TCP/UDP/IP probes; one-shot fingerprinting |
| Service Discovery | Nmap | Reconnaissance | FTP/DNS/Web servers | TCP/UDP probes; one-shot version scan |
| Advanced Discovery | Nmap | Reconnaissance | ONOS controller | TCP/UDP/IP probes; one-shot combined scan |
| Full Port Scan | Nmap | Reconnaissance | All subnets | TCP scan; full-port sweep |
| Full host discovery | Nmap | Reconnaissance | All devices | ICMP/IP probes; one-shot wide sweep |
| SSH Brute Force | Hydra | Credential guessing | TCP 22/SSH | Repeated SSH logins |
| HTTP Brute Force | Hydra | Credential guessing | HTTP Server | Repeated HTTP request attempts |
| Brute Force SSH & Web | Hydra mixed | Credential guessing | SSH and web login | Repeated SSH/HTTP attempts |
| Slowloris HTTP | Python socket | Resource exhaustion | TCP 80/HTTP | 100 sockets; header refresh every 15 s |
| HTTP DoS and SYN Flood | hping3 and wrapper | Resource exhaustion | TCP 80/HTTP | Concurrent TCP floods; host-limited rate |
| DoS against OpenFlow Channel | hping3 | Resource exhaustion | ONOS, TCP 6653 | Continuous, host-limited rate |
| DDoS SYN Flood | hping3 | Resource exhaustion | ONOS, TCP 6653 | Continuous, host-limited rate |
| DDoS UDP Flood | hping3 | Resource exhaustion | ONOS, UDP 6653 | Continuous, host-limited rate |

*Continued on the next page*

| Attack label | Tool | Target semantics | Endpoint | Primitive and send pattern |
|---|---|---|---|---|
| Coordinated DDoS | hping3 and wrapper | Resource exhaustion | ONOS and OVS, TCP 6653 | Parallel TCP floods |
| DNS Amplification | dig | Amplification | 30 DNS queries per execution | |
| LDAP Amplification | Python socket | Amplification | UDP 389/LDAP | UDP datagram every 0.5 s |
| Smurf Attack | Scapy | Reflection | Broadcast ICMP | 50 echo requests every 5 s |
| Packet Crafting OpenFlow | Scapy | Injection | ONOS, TCP 6653 | Ether/IP/TCP + raw payload; 1 pkt/s for 2 h |
| OpenFlow Traffic Injection | Scapy | Injection | OVS, TCP 6653 | Ether/IP/TCP + raw payload; 20-packet bursts |
| Flow Table Flooding | Scapy | Resource exhaustion | ONOS/OVS, TCP 6653 | Ether/IP/TCP + raw payload; 100-packet bursts |
| Flooding Flow Rules | Scapy | Resource exhaustion | OVS, TCP 6653 | Ether/IP/TCP + raw payload; 200-packet bursts |

Overall, the attack scenarios in KRONOS-SDN provide a reproducible characterization of malicious traffic across multiple stages of the experimental campaign, including reconnaissance, credential guessing, application-layer denial of service, amplification-oriented abuse, volumetric flooding and OpenFlow-endpoint stress. By making explicit the packet-generation layer, the destination endpoints and the nominal send patterns, this appendix supports a clearer comparison with other IDS datasets, including those that model SDN attacks at different levels of abstraction or rely on explicit OpenFlow message construction.

# Appendix C

# KRONOS-SDN SHAP Analysis

This appendix complements the discussion in Section 4.7 and the representative SVM explanation reported in Figure 12 by providing SHAP summary beeswarm plots for the other classifiers. Figures C.4, C.5 and C.6 aggregate the explanations for the six models, that is to say Naïve Bayes, Decision Tree, Random Forest, AdaBoost, MLP and k-NN, on the same KRONOS-SDN setting, consisting of ONOS VM with binary classification. In each panel, features are ranked by mean absolute SHAP value; the x-axis reports the signed contribution to the model output for the positive class and the color encodes the feature magnitude (low in blue, high in red). The last row groups the *sum of the remaining (non-displayed) features*, offering a compact view of how much predictive evidence is carried by the feature long tail.

**Cross-model behavior.** The six models exhibit a recurrent set of highly-informative cues among homogeneous model families. Timing descriptors (e.g., inter-arrival-time statistics and flow duration) appear prominently in the tree-based and boosted models, while packet-length dispersion statistics (e.g., packet length standard deviation and direction-dependent packet-length moments) remain influential across

Naïve Bayes, Random Forest, MLP and k-NN. In addition, transport-state indicators, that is to say TCP flags and initial window bytes, emerge as top drivers for MLP and k-NN, suggesting that deviations in stateful TCP behavior and stack-level characteristics provide complementary evidence to pure timing and size-based descriptors.

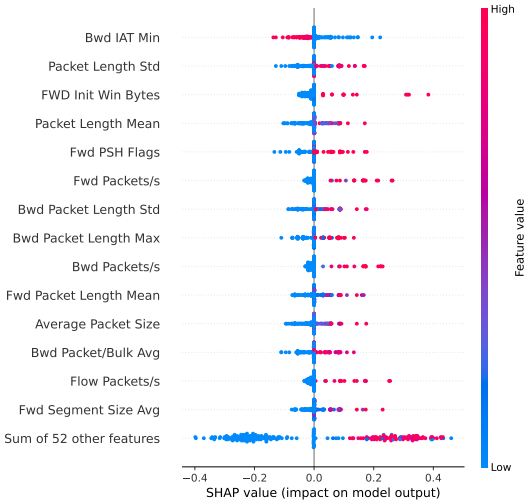
**Model-specific attribution structure.** The Naïve Bayes summary in Figure C.4a) highlights a combination of packet-length variability and rate-related descriptors, consistent with a classifier that relies on marginal distribution shifts in multiple partially informative variables rather than on a single dominant separator.

The Decision Tree shown in Figure C.4b) concentrates its attributions on a small subset of timing and duration features, reflecting a sparse decision logic driven by few high-impact splits. Moving to ensembles, Random Forest (Figure C.5a) spreads the importance over a broader set of descriptors, including timing, rates and packet-length variability, which is coherent with the averaging of many weak rules that capture different aspects of the attack/benign separation. AdaBoost (Figure C.5b) shows a strong contribution from timing minima together with additional flag- and duration-related cues, consistent with boosted combinations of simple threshold-based learners that amplify discriminative timing artefacts while still leveraging secondary signals.

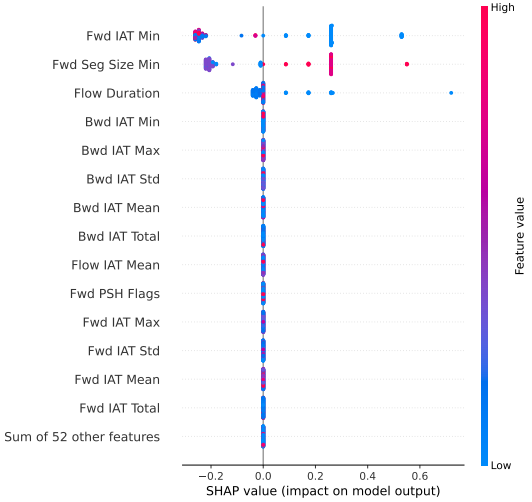
The MLP shown in Figure C.6a and k-NN in Figure C.6b emphasize transport-state and connection-setup characteristics (e.g., initial window bytes and flag counts), alongside packet-length dispersion and direction-dependent statistics. This indicates that non-linear and distance-based learners may exploit multi-dimensional similarity patterns where attacks differ from benign traffic not only in aggregate timing/rate properties but also in protocol-state dynamics and distributional variability.

**Long-tail evidence and implications.** Across all six panels, the aggregated *sum of the remaining features* exhibits a non-negligible and widely distributed contribution. This corroborates the main-text observation that predictive evidence is not concentrated in a handful of trivially ma-

nipulable variables: even when a few features show clear separation patterns, a substantial fraction of the decision signal is carried by many additional descriptors whose cumulative effect is significant. From a security perspective, this attribution spread supports a more robust separation, since evasion would require coordinated mimicry of benign behavior across heterogeneous dimensions (timing regularity, stateful transport artefacts and packet-size dispersion), rather than simple traffic shaping on a single dominant statistic.

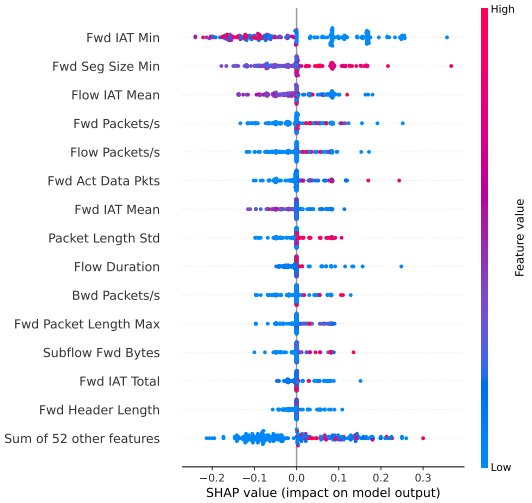



**(a)** Naïve Bayes

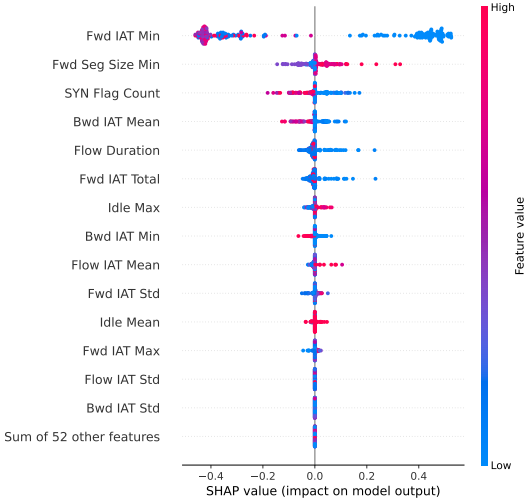



**(b)** Decision Tree

**Figure C.4:** Beeswarm SHAP plots for DDoS attack in KRONOS dataset for NB and DT.

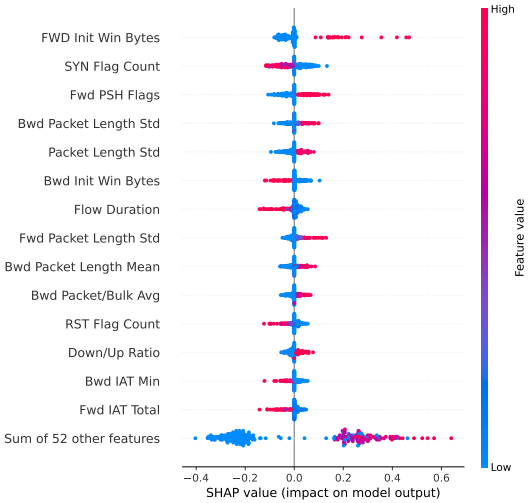



**(a)** Random Forest

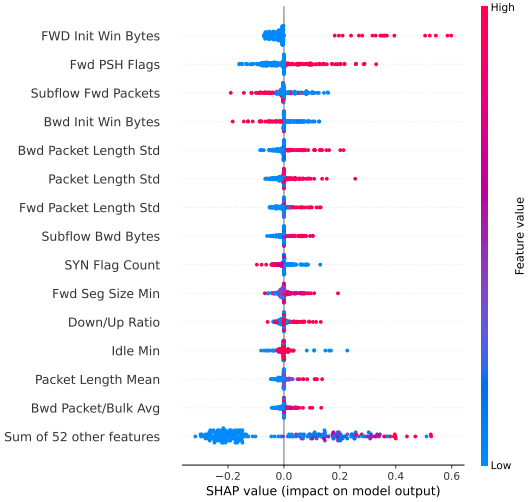



**(b)** AdaBoost

**Figure C.5:** Beeswarm SHAP plots for DDoS attack in KRONOS dataset for RF and AdaBoost.




**(a)** MLP




**(b)** k-NN

**Figure C.6:** Beeswarm SHAP plots for DDoS attack in KRONOS dataset for MLP and k-NN.

# Appendix D

# Context Severity and Level for ONOS VM

Building on the discussion in Subsection 6.8.8, this appendix provides the complete set of daily context traces for the entire experimental week. For each day, the figures report the evolution of the continuous context severity signal ($ctx\_severity$) and its discretized operating-regime indicator ($ctx\_level$), together with the ground-truth attack intervals.

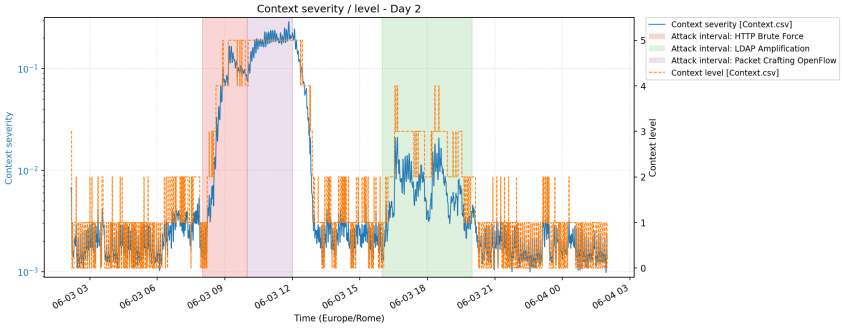
The following aims guided this discussion: first, to document the day-by-day variability of controller-side stress responses across heterogeneous attack types and traffic conditions; second, to visually corroborate the key behaviors highlighted in the main text, transitions transitions to high-severity regimes during controller-targeting attacks, heterogeneous amplitude and persistence depending on the specific attack interval. Moreover, focal aspects consist of the non-negligible recovery latencies in which the controller remains in a perturbed regime after the malicious activity ends. Collectively, these traces offer additional evidence that $ctx\_level(t)$ captures sustained operating-regime shifts and therefore constitutes a meaningful control signal to modulate the traffic-side decision threshold $\tau(t)$ in the context-aware coupling proposed in this work.

Day 2 (Figure D.7a) shows a more heterogeneous behavior: the first

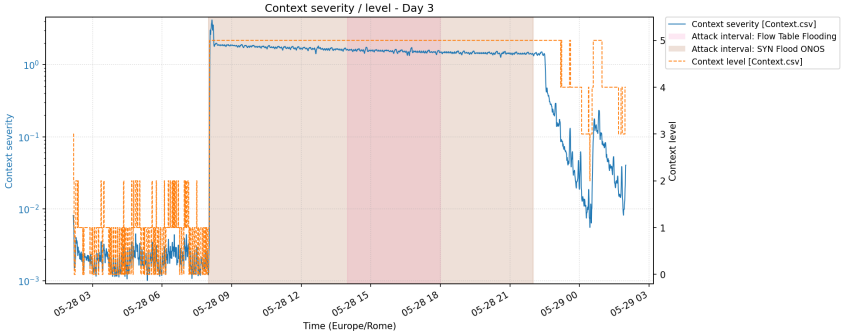
attack window triggers a clear ramp-up of $ctx\_severity$ and elevated context levels, whereas the subsequent interval produces a weaker (or less persistent) response, indicating that not all attack types exert the same pressure on the controller host telemetry. In Day 3 (Figure D.7b), the effect is stronger and more structured: $ctx\_severity$ undergoes an abrupt transition to a high-severity regime that persists for an extended period, with $ctx\_level$ remaining saturated at upper levels throughout most of the malicious window; the return to normal conditions occurs only later, again evidencing a non-negligible recovery time.

Day 4 (Figure D.8a) serves as a useful reference baseline: in the absence of major attacks (or under low-impact activity), $ctx\_severity$ remains close to its nominal range and $ctx\_level$ fluctuates mostly among the lowest bins, reflecting ordinary workload variability rather than sustained stress. Day 5 (Figure D.8b) exhibits the opposite pattern: a multi-attack period yields a rapid onset followed by a prolonged elevation of $ctx\_severity$, with $ctx\_level$ staying high for most of the shaded interval. Importantly, after the end of the attacks the signal decays slowly, implying that the controller host state remains affected for a while (e.g., due to residual load, queued work, or lingering network-side effects). A similar but even more pronounced phenomenon is visible on Day 6 (Figure D.9a), where attack phases induce both sharp peaks and long high-severity segments; the post-attack tail is particularly evident, with $ctx\_severity$ decreasing progressively rather than collapsing immediately to baseline.

Finally, Day 7 (Figure D.9b) aggregates multiple attack intervals and shows an extended high-severity regime with repeated local peaks, consistent with a prolonged period of abnormal controller-side activity. As in previous days, the recovery is not instantaneous: $ctx\_level$ remains elevated beyond the end of the last shaded interval and only later returns to low bins. Overall, these daily traces support the rationale for context-aware coupling: since $ctx\_level(t)$ tracks persistent operating-regime shifts and delayed recovery phases, it provides a meaningful control signal for dynamically selecting the traffic-side decision threshold $\tau(t)$ in a way that is aligned with the controller's current condition.

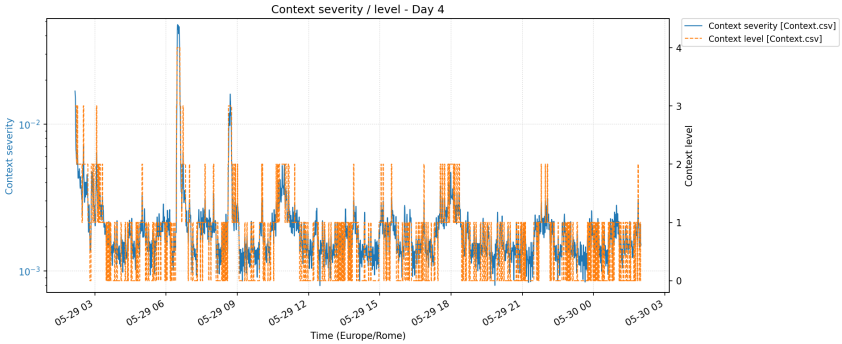



**(a)** Day 2

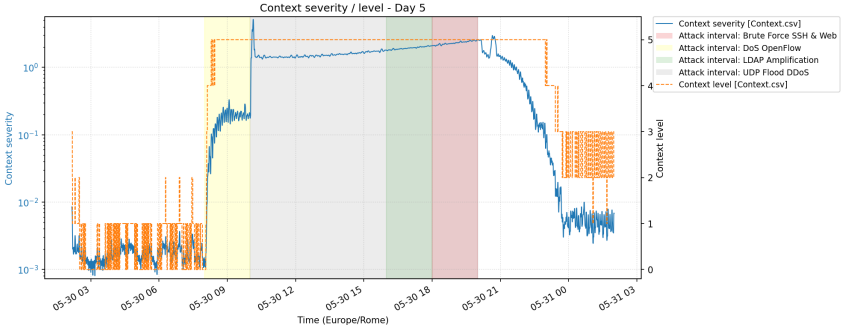



**(b)** Day 3

**Figure D.7:** Context severity and discretized level for Days 2-3.

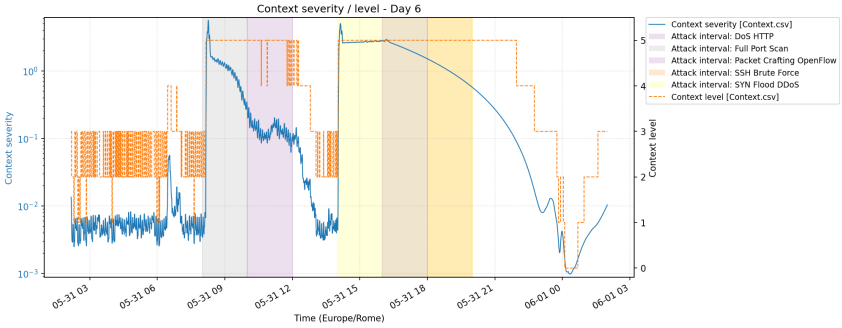



**(a)** Day 4

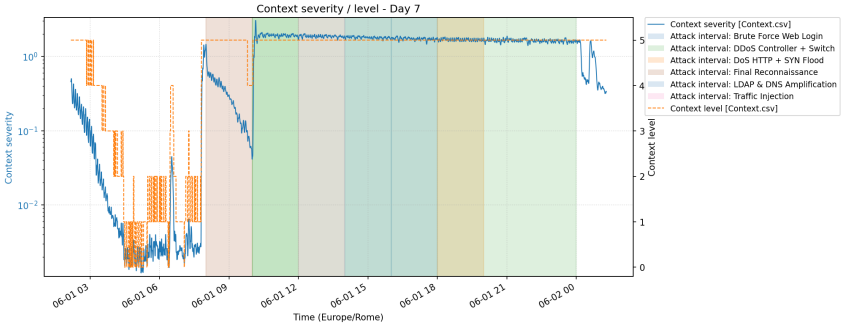



**(b)** Day 5

**Figure D.8:** Context severity and discretized level for Days 4-5.




**(a)** Day 6




**(b)** Day 7

**Figure D.9:** Context severity and discretized level for Days 6-7.

# Bibliography

*A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018)* (2018). Registry of Open Data on AWS (managed by Canadian Institute for Cybersecurity). URL: https://registry.opendata.aws/cse-cic-ids2018/ (visited on 12/13/2025).

Abderrahmane, Amel, Hamza Drid, and Amel Behaz (2024). "A Survey of Controller Placement Problem in SDN-IoT Network". In: *International Journal of Networked and Distributed Computing* 12, pp. 170–184. DOI: 10.1007/s44227-024-00035-y.

Ahuja, Nisha, Gaurav Singal, and Debajyoti Mukhopadhyay (Sept. 2020). *DDOS attack SDN Dataset*. Version 1. Accessed: 2025-12-12. DOI: 10.17632/jxpfjc64kr.1. URL: https://data.mendeley.com/datasets/jxpfjc64kr/1.

AL, Samed and Seref Sagiroglu (2025). "Explainable artificial intelligence models in intrusion detection systems". In: *Engineering Applications of Artificial Intelligence* 144, p. 110145. DOI: 10.1016/j.engappai.2025.110145.

Aladaileh, Mohammad Adnan, Mohammed Anbar, Ahmed J. Hintaw, Iznan H. Hasbullah, Abdullah Ahmed Bahashwan, and Shadi Al-Sarawi (2022). "Renyi Joint Entropy-Based Dynamic Threshold Approach to Detect DDoS Attacks Against SDN Controller with Various Traffic Rates". In: *Applied Sciences* 12.12, p. 6127. DOI: 10.3390/app12126127.

Aldweesh, Areej, Abdelouahid Derhab, and A. Z. Emam (2020). "Deep Learning Approaches for Anomaly-Based Intrusion Detection Systems: A Survey, Taxonomy, and Open Issues". In: *Knowledge-Based Systems* 189, p. 105124. DOI: 10.1016/j.knosys.2019.105124.

Ali, A. et al. (2023). "Deep Learning Based DDoS Detection in SDN: A Survey". In: *Computers and Electrical Engineering* 110, p. 108734. DOI: 10.1016/j.compeleceng.2023.108734.

Alkasassbeh, Mouhanad and Muneer Al-Haj Baddar (2023). "Intrusion Detection Systems: A State-of-the-Art Taxonomy and Survey". In: *Arabian Journal for Science and Engineering*.

Alotaibi, Afnan and Murad A. Rassam (2023). "Adversarial Machine Learning Attacks against Intrusion Detection Systems: A Survey on Strategies and Defense". In: *Future Internet* 15.2, p. 62. DOI: 10.3390/fi15020062.

Aoudi, Wissam and Magnus Almgren (2021a). "A Framework for Determining Robust Context-Aware Attack-Detection Thresholds for Cyber-Physical Systems". In: *Proceedings of the 2021 Australasian Computer Science Week Multiconference (ACSW '21)*. Dunedin, New Zealand: ACM, pp. 1–6. DOI: 10.1145/3437378.3437393.

— (2021b). "A Framework for Determining Robust Context-Aware Attack-Detection Thresholds for Cyber-Physical Systems". In: *Proceedings of the 2021 Australasian Computer Science Week Multiconference*. ACSW '21. Dunedin, New Zealand: Association for Computing Machinery. ISBN: 9781450389563. DOI: 10.1145/3437378.3437393. URL: https://doi.org/10.1145/3437378.3437393.

Assis, M. V. O., A. H. Hamamoto, T. Abrao, and M. L. Proença (2020). "SDN Enabled IoT Security: A CNN-Based Intrusion Detection Approach". In: *Computer Networks*.

— (2021). "A GRU-Based Intrusion Detection System for SDN". In: *Computer Networks*.

Ataa, M. Sami, Eman E. Sanad, and Reda A. El-khoribi (Nov. 2024a). "Intrusion detection in software defined network using deep learning approaches". In: *Scientific Reports* 14.1, p. 29159. DOI: 10.1038/s41598-024-79001-1.

— (2024b). "Intrusion detection in software defined network using deep learning approaches". In: *Scientific Reports* 14. Article number: 29159. DOI: 10.1038/s41598-024-79001-1.

Bahashwan, A., Mohammed Anbar, and Iznan H. Hasbullah (2024). "HLD-DDoSDN: High and Low-Rates Dataset-Based DDoS Attacks Against SDN". In: *PLOS ONE* 19.2, e0297548. DOI: 10.1371/journal.pone.0297548.

Bahashwan, Abdullah A., Mohammed Anbar, Subramaniam Manickam, Taher A. Al-Amiedy, Mohammad Adnan Aladaileh, and Iznan H. Hasbullah (2023). "A Systematic Literature Review on Machine Learn-

ing and Deep Learning Approaches for Detecting DDoS Attacks in Software-Defined Networking". In: *Sensors* 23.9, p. 4441. DOI: 10.3390/s23094441.

Batool, Sidra, Muhammad Aslam, Edore Akpokodje, and Syeda Fizzah Jilani (2024). "A comprehensive plane-wise review of DDoS attacks in SDN: Leveraging detection and mitigation through machine learning and deep learning". In: *Journal of Network and Computer Applications*. URL: https://www.sciencedirect.com/science/article/abs/pii/S1084804524002583.

Borgioli, A. et al. (2024). "Embedded Context-Aware Autoencoders for Industrial Anomaly Detection". In: *IEEE Internet of Things Journal*.

Brooks, Michael and Jie Yang (2015). "A Man-in-the-Middle Attack against OpenDayLight SDN Controller using ARP Poisoning". In: *Proceedings of the 4th Annual ACM Conference on Research in Information Technology (RIIT)*. DOI: 10.1145/2808062.2808073.

Cevallos, Jesus, Alessandra Rizzardi, Sabrina Sicari, and Alberto Coen-Porisini (Sept. 2024). "ASAP: Automatic Synthesis of Attack Prototypes, an online-learning, end-to-end approach". In: *Computer Networks* 254, p. 110828. DOI: 10.1016/j.comnet.2024.110828.

Chalapathy, Raghavendra and Sanjay Chawla (2019). "Deep Learning for Anomaly Detection: A Survey". In: *arXiv preprint*. arXiv:1901.03407. DOI: 10.48550/arXiv.1901.03407.

Chen, Ye et al. (2018). "A DDoS Detection Method for SDN Based on XGBoost". In: *Proceedings of the 2018 International Conference on Communications, Signal Processing, and Systems*, pp. 1241–1251.

Chica, Juan C., José C. Imbachi, and Juan F. Botero (2020). "Security in SDN: A Comprehensive Survey". In: *Journal of Network and Computer Applications* 159, p. 102595. DOI: 10.1016/j.jnca.2020.102595.

Clausen, A. et al. (2021). "Context-Based Anomaly Mitigation for Streaming Detection". In: *IEEE Access*.

collectd Project (2025). *collectd: The system statistics collection daemon*. Project website. URL: https://collectd.org/.

Cutkosky, Ashok, Aaron Defazio, and Harsh Mehta (2023). "Mechanic: A Learning Rate Tuner". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

David, Olubukola, Paul Thornley, and Maryam Bagheri (2023). "Software Defined Networking (SDN) for Campus Networks, WAN, and Datacenter". In: *2023 International Conference on Smart Applications, Communications and Networking (SmartNets)*, pp. 1–8. DOI: 10.1109/SmartNets58706.2023.10215722.

Dawoud, A., S. Shahristani, and C. Raun (2019). "Deep Learning Approach for Anomaly Detection in SDN". In: *2019 International Conference on Computer and Information Sciences (ICCIS)*, pp. 1–6.

Deb, Raktim and Sudipta Roy (2022). "A comprehensive survey of vulnerability and information security in SDN". In: *Computer Networks* 206, p. 108802. DOI: 10.1016/j.comnet.2022.108802.

Deng, Shuhua, Xing Gao, Zebin Lu, and Xieping Gao (2018). "Packet Injection Attack and Its Defense in Software-Defined Networks". In: *IEEE Transactions on Information Forensics and Security* 13.3, pp. 695–705. DOI: 10.1109/TIFS.2017.2765506.

Dhirar, Heba and Ali Hamad (2025). "Comparative Evaluation of a Novel IDS Dataset for SDN-IoT Using Deep Learning Models Against InSDN, BoT-IoT, and ToN-IoT". In: *Machine Learning with Applications* 16, p. 100015. DOI: 10.1016/j.mlwa.2025.100015.

Di Gennaro, Francesco et al. (2024). *Feature selection study on InSDN with benchmark ML models and SHAP analysis*. © 2024 IEEE.

Di Gennaro, Francesco, Alessandro Cucchiarelli, Christian Morbidoni, and Luca Spalazzi (2025). "A Comparative Analysis of Datasets for Intrusion Detection in Software-Defined Networks". In: *Proceedings of the Joint Italian Conference on CyberSecurity (ITASEC & SERICS 2025)*. Vol. 3962. CEUR Workshop Proceedings. CEUR-WS.org. URL: https://ceur-ws.org/Vol-3962/.

Ebrahem, E. et al. (2025). "Towards a minimum universal features set for IoT DDoS attack". In: *Journal of Big Data* 12.1, p. 46. DOI: 10.1186/s40537-025-01146-1.

Elsayed, Mahmoud Said, Nhien-An Le-Khac, and Anca D. Jurcut (2020). "InSDN: A Novel SDN Intrusion Dataset". In: *IEEE Access* 8, pp. 165263–165284. DOI: 10.1109/ACCESS.2020.3022633.

European Union Agency for Cybersecurity (ENISA) (Sept. 2024). *ENISA Threat Landscape 2024*. Tech. rep. ENISA. DOI: 10.2824/071088. URL: https://securitydelta.nl/media/com_hsd/report/690/document/ENISA-Threat-Landscape-2024.pdf.

Ferrag, Mohamed Amine, Leandros Maglaras, Sotiris Moschoyiannis, and Helge Janicke (2020). "Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study". In: *Journal of Information Security and Applications* 50, p. 102419. ISSN: 2214-2126. DOI: https://doi.org/10.1016/j.jisa.2019.102419. URL: https://www.sciencedirect.com/science/article/pii/S2214212619305046.

Griffin, D. et al. (2020). "An Ensemble-Based Deep Learning Approach for Intrusion Detection in SDN". In: *IEEE Access*.

Guo, G., H. Wang, D. Bell, Y. Bi, and K. Greer (2003). "KNN Model-Based Approach in Classification". In: *On The Move to Meaningful Internet Systems 2003: OTM Workshops*. Springer, pp. 986–996.

Gyanchandani, Manoj, Jyoti Lal Rana, and Ramesh N. Yadav (2012). "Taxonomy of Anomaly Based Intrusion Detection System: A Review". In: *International Journal of Scientific and Research Publications* 2.12.

Haider, S. et al. (2020). "An Ensemble of CNN Models for DDoS Detection in SDN". In: *Future Generation Computer Systems*.

Han, Biao, Xiangrui Yang, Zhigang Sun, Jinfeng Huang, and Jinshu Su (Jan. 2018). "OverWatch: A Cross-Plane DDoS Attack Defense Framework with Collaborative Intelligence in SDN". In: *Security and Communication Networks* 2018, pp. 1–15. DOI: 10.1155/2018/9649643.

Hande, S., A. Muddana, and S. Bakshi (2020). "A Survey on Intrusion Detection System for Software Defined Networks (SDN)". In: *International Journal of Business Data Communications and Networking* 16.4. DOI: 10.4018/IJBDCN.2020100106.

Health Sector Cybersecurity Coordination Center (HC3) (Sept. 2024). *DDoS Attacks: A Guide for Healthcare*. Tech. rep. U.S. Department of Health and Human Services. URL: https://www.hhs.gov/sites/default/files/ddos-attacks-guide-for-healthcare.pdf.

Hearst, Marti A., Susan T. Dumais, Edgar Osuna, John Platt, and Bernhard Schölkopf (1998). "Support Vector Machines". In: *IEEE Intelligent Systems* 13.4, pp. 18–28.

Hindy, Hanan, David Brosset, Elliott Bayne, Amar Seeam, Christos Tachtatzis, Robert Atkinson, and Xavier Bellekens (2018). "A Taxonomy and Survey of Intrusion Detection System Design Techniques, Network-Based Intrusion Detection Datasets and Challenges". In: *arXiv preprint*. arXiv: 1806.03517.

Hinton, Geoffrey E. and Ruslan R. Salakhutdinov (2006). "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786, pp. 504–507. DOI: 10.1126/science.1127647.

Hong, Seungyeop, Lei Xu, Haopei Wang, and Guofei Gu (2015). "Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures". In: *Network and Distributed System Security Symposium (NDSS)*. DOI: 10.14722/ndss.2015.23283.

Howe, Alex, Andrew Morin, Mauricio Papa, and Tyler Moore (2025). "A Cost-Sensitive Approach for Managing Intrusion Alerts in OT En-

vironments". In: *Critical Information Infrastructures Security (CRITIS 2024), Revised Selected Papers*. Vol. 15549. Lecture Notes in Computer Science. Springer, Cham, pp. 306–325. DOI: 10.1007/978-3-031-84260-3_18.

Jafarian, Jalil et al. (2020). "A Survey on Anomaly Detection in SDN: Techniques, Challenges and Future Directions". In: *Computers & Security* 97, p. 101993. DOI: 10.1016/j.cose.2020.101993.

Janabi, M. et al. (2024). "Intrusion Detection in Software-Defined Networks: A Survey". In: *IEEE Access*.

Jero, Samuel, Xiangyu Bu, Cristina Nita-Rotaru, Hamed Okhravi, Richard Skowyra, and Sonia Fahmy (2017). "BEADS: Automated Attack Discovery in OpenFlow-Based SDN Systems". In: *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*, pp. 311–333. DOI: 10.1007/978-3-319-66332-6_14.

Khalid, H. Y. I. et al. (2024). "A survey on the latest intrusion detection datasets for software defined networking environments". In: *Engineering, Technology & Applied Science Research* 14.2, pp. 13190–13200. DOI: 10.48084/etasr.6756.

Khanal, Bipal, Chandan Kumar, and Md. Sarfaraj Alam Ansari (2023). "NITSDN: Development of SDN Dataset for ML-Based Intrusion Detection System". In: *Advanced Computational and Communication Paradigms (ICACCP 2023)*. Vol. 535. Lecture Notes in Networks and Systems. Singapore: Springer, pp. 99–111. DOI: 10.1007/978-981-99-4284-8_8.

Khraisat, Ansam, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman (2019). "Survey of Intrusion Detection Systems: Techniques, Datasets and Challenges". In: *Cybersecurity* 2.20. DOI: 10.1186/s42400-019-0038-7.

Kissell, Robert and Jim Poserina (2017). "Chapter 2 - Regression Models". In: *Optimal Sports Math, Statistics, and Fantasy*. Ed. by Robert Kissell and Jim Poserina. Academic Press, pp. 39–67. ISBN: 978-0-12-805163-4. DOI: https://doi.org/10.1016/B978-0-12-805163-4.00002-5. URL: https://www.sciencedirect.com/science/article/pii/B9780128051634000025.

Kreutz, D., Fernando M. V. Ramos, Paulo Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig (2014). "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1, pp. 14–76. DOI: 10.1109/JPROC.2014.2371999.

Kreutz, Diego, Fernando M. V. Ramos, Paulo E. Veríssimo, Christian E. Rothenberg, Sina Azodolmolky, and Steve Uhlig (2015). "Software-

Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1, pp. 14–76. DOI: 10.1109/JPROC.2014.2371999.

Latah, M. and L. Toker (2018). "Towards a Multilevel Intrusion Detection System for SDN". In: *IEEE Access* 6, pp. 74217–74228.

Lazarevic, Aleksandar, Vipin Kumar, and Jaideep Srivastava (2005). "Intrusion Detection: A Survey". In: *Managing Cyber Threats: Issues, Approaches and Challenges*. Ed. by Vipin Kumar, Jaideep Srivastava, and Aleksandar Lazarevic. Springer, pp. 19–78.

Leivadeas, Aris and Matthias Falkner (2023). "A Survey on Intent-Based Networking". In: *IEEE Communications Surveys & Tutorials* 25.1, pp. 625–655. DOI: 10.1109/COMST.2022.3215919.

Li, Shihua et al. (2018). "Software-Defined Network-Based DDoS Detection and Mitigation: A Survey". In: *Future Internet* 10.4.

Liao, Ping-Sung, Tse-Sheng Chen, and Pau-Choo Chung (2001). "A Fast Algorithm for Multilevel Thresholding". In: *Journal of Information Science and Engineering* 17.5, pp. 713–727.

Liu, Hong and Bo Lang (2019). "Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey". In: *Applied Sciences* 9.20, p. 4396. DOI: 10.3390/app9204396.

Liu, Sheng, Michael K. Reiter, and Vyas Sekar (2017). "Flow Reconnaissance via Timing Attacks on SDN Switches". In: *37th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 196–206. DOI: 10.1109/ICDCS.2017.281.

Lumen Black Lotus Labs (2022). *Connection-less LDAP (CLDAP) can be abused for DDoS*. Lumen / Black Lotus Labs blog. URL: https://blog.lumen.com/connection-less-ldap-cldap-can-be-abused-for-ddos/.

Lundberg, Scott M. and Su-In Lee (2017). "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30 (NeurIPS)*, pp. 4765–4774.

Marcílio, William E. and Daniel M. Eler (2020). "From Explanations to Feature Selection: Assessing SHAP Values as Feature Selection Mechanism". In: *33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pp. 340–347. DOI: 10.1109/SIBGRAPI51738.2020.00053.

Melis, Andrea, Amir Al Sadi, Davide Berardi, Franco Callegati, and Marco Prandini (2023). "A Systematic Literature Review of Offensive and Defensive Security Solutions With Software Defined Network". In: *IEEE Access* 11, pp. 93431–93463. DOI: 10.1109/ACCESS.2023.3276238.

Mirsky, Yisroel, Tamar Doitshman, Yuval Elovici, and Asaf Shabtai (2018). "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection". In: *Network and Distributed Systems Security Symposium (NDSS)*.

Mohale, Vincent Zibi and Ibidun Christiana Obagbuwa (2025). "A systematic review on the integration of explainable artificial intelligence in intrusion detection systems to enhancing transparency and interpretability in cybersecurity". In: *Frontiers in Artificial Intelligence* 8, p. 1526221. DOI: 10.3389/frai.2025.1526221.

Moustafa, Nour and Jill Slay (2015). "UNSW-NB15: A Comprehensive Data Set for Network Intrusion Detection Systems (UNSW-NB15 Network Data Set)". In: *2015 Military Communications and Information Systems Conference (MilCIS)*, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942.

Mustafa, Zaid, Rashid Amin, Hamza Aldabbas, and Naeem Ahmed (Oct. 2024). "Intrusion detection systems for software-defined networks: a comprehensive study on machine learning-based techniques". In: *Cluster Computing* 27.7, pp. 9635–9661. DOI: 10.1007/s10586-024-04430-6.

Naeen, Hossein Monshizadeh, Malihe Ghadamyari, and Mobin Barmar (2025). "Enhancing SDN security with deep learning and F-balanced cross-entropy for DDoS detection". In: *Scientific Reports* 15, p. 33419. DOI: 10.1038/s41598-025-18826-w.

Niyaz, Q., Weiqing Sun, and Ahmad Y. Javaid (2017). "A Deep Learning Based DDoS Detection System in Software-Defined Networking (SDN)". In: *EAI Endorsed Transactions on Security and Safety* 4.12, e2. DOI: 10.4108/eai.28-12-2017.153515.

Niyaz, Q., Weiqing Sun, and Ayaz Javaid (2017). "A Deep Learning Based DDoS Detection System in Software-Defined Networking (SDN)". In: *arXiv preprint*. arXiv: 1611.07400.

Novaes, Matheus P., Luiz Fernando Carvalho, Jaime Lloret, and Mario Lemes Proença Jr. (2020). "Long Short-Term Memory and Fuzzy Logic for Anomaly Detection and Mitigation in Software-Defined Network Environment". In: *IEEE Access* 8, pp. 83765–83781. DOI: 10.1109/ACCESS.2020.2992044.

Open Networking Foundation (2014a). *SDN Architecture*. Tech. rep. TR-502. ONF. URL: https://opennetworking.org.

— (2014b). *SDN Architecture*. Tech. rep. TR-502. ONF. URL: https://opennetworking.org.

— (2015). *OpenFlow Switch Specification, Version 1.3.5*. Tech. rep. ONF TS-025. Open Networking Foundation. URL: https://opennetworking.org.

Ortega-Fernández, A. et al. (2022). "Autoencoder-Based Anomaly Detection for Industrial Control Systems". In: *Computers & Security*.

Otsu, Nobuyuki (1979). "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1, pp. 62–66. DOI: 10.1109/TSMC.1979.4310076.

Parmar, Amit, Rohit Katariya, and Vishal Patel (2019). "A Review on Random Forest: An Ensemble Classifier". In: *International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI 2018)*. Springer, pp. 758–763.

Pascoal, Túlio A., Yuri Gil Dantas, Iguatemi E. Fonseca, and Vivek Nigam (2017). "Slow TCAM Exhaustion DDoS Attack". In: *ICT Systems Security and Privacy Protection (IFIP SEC)*, pp. 17–31. DOI: 10.1007/978-3-319-58469-0_2.

Pawlicki, Marek, Aleksandra Pawlicka, Rafal Kozik, and Michal Choraś (2025). "A meta-survey of adversarial attacks against artificial intelligence algorithms, including diffusion models". In: *Neurocomputing*, p. 131231. DOI: 10.1016/j.neucom.2025.131231.

Pittman, Jason M. (2023). "Machine Learning and Port Scans: A Systematic Review". In: arXiv:2301.13581v1. arXiv: 2301.13581 [cs.CR].

Rana, Md Shafiur, Md Shazzadul Haque, and Md Shafayet Rahman (2022). "An Efficient Feature Selection Approach for Intrusion Detection". In: *International Journal of Advanced Computer Science and Applications (IJACSA)* 13.6, pp. 531–538. DOI: 10.14569/IJACSA.2022.0130663. URL: https://scispace.com/pdf/an-efficient-feature-selection-approach-for-intrusion-17ev2w9q.pdf.

Rauf, Bilal, Haider Abbas, Muhammad Usman, Tanveer A. Zia, Waqas Iqbal, Yasir Abbas, and Hasan Afzal (2021). "Application Threats to Exploit Northbound Interface Vulnerabilities in Software Defined Networks". In: *ACM Computing Surveys* 54.6, pp. 1–36. DOI: 10.1145/3453648.

Ring, Markus, Stefan Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho (2019). "A Survey of Network-Based Intrusion Detection Data Sets". In: *Computers & Security* 86, pp. 147–167. DOI: 10.1016/j.cose.2019.06.005.

Rish, Irina (2001). "An Empirical Study of the Naive Bayes Classifier". In: *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*.

Sakurada, Mayu and Takehisa Yairi (2014). "Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction". In: *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. MLSDA'14. Gold Coast, Australia QLD, Australia: Association for Computing Machinery, pp. 4–11. ISBN: 9781450331593. DOI: 10.1145/2689746.2689747. URL: https://doi.org/10.1145/2689746.2689747.

Sarhan, M. (2024). "Feature extraction for machine learning-based intrusion detection". In: *Journal of Information Security and Applications* 76, p. 103676. DOI: 10.1016/j.jisa.2022.103676.

Sarica, Alper Kaan and Pelin Angin (2020). "A Novel SDN Dataset for Intrusion Detection in IoT Networks". In: *2020 16th International Conference on Network and Service Management (CNSM)*, pp. 1–5. DOI: 10.23919/CNSM50824.2020.9269042.

Scott-Hayward, Sandra, Sriram Natarajan, and Sakir Sezer (2016). "A Survey of Security in Software Defined Networks". In: *IEEE Communications Surveys & Tutorials* 18.1, pp. 623–654.

Sharafaldin, Iman, Arash Habibi Lashkari, and Ali A. Ghorbani (2018a). "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP)*, pp. 108–116. DOI: 10.5220/0006639801080116.

— (2018b). "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP)*, pp. 108–116. DOI: 10.5220/0006639801080116.

Shen, Yi, Guangjie Han, Yuying Sun, Xinheng Liu, and Mohamed Guizani (2023). "Flow Table Saturation Attack against Dynamic Timeout Mechanisms in SDN". In: *Applied Sciences* 13.12, p. 7243. DOI: 10.3390/app13127243. URL: https://www.mdpi.com/2076-3417/13/12/7243.

Shin, Seungwon, Phillip Porras, Vinod Yegneswaran, and Guofei Gu (2014). "Rosemary: A Robust, Secure, and High-Performance Network Operating System". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. DOI: 10.1145/2660267.2660353.

Shyaa, Methaq A., Noor Farizah Ibrahim, Zurinahni Zainol, Rosni Abdullah, Mohammed Anbar, and Laith Alzubaidi (2024). "Evolving cybersecurity frontiers: A comprehensive survey on concept drift and feature dynamics aware machine and deep learning in intrusion de-

tection systems". In: *Engineering Applications of Artificial Intelligence* 137, p. 109143. DOI: 10.1016/j.engappai.2024.109143.

Singh, Gurpreet and Madan Sachan (2014). "Multi-layer Perceptron Neural Network Technique for Offline Handwritten Gurmukhi Character Recognition". In: *2014 IEEE International Conference on Computational Intelligence and Computing Research*, pp. 1–6. DOI: 10.1109/ICCIC.2014.7238391.

Singh, Sachin Kumar, Shreeman Gautam, Cameron Cartier, Sameer Patil, and Robert Ricci (Apr. 2024). "Where The Wild Things Are: Brute-Force SSH Attacks In The Wild And How To Stop Them". In: *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI '24)*. Santa Clara, CA, USA, pp. 1731–1751. URL: https://www.usenix.org/system/files/nsdi24-singh-sachin.pdf.

Song, Yan and Ying Lu (2015). "Decision Tree Methods: Applications for Classification". In: *Shanghai Archives of Psychiatry* 27.2, pp. 130–135.

Stolfo, Salvatore, Wei Fan, Wenke Lee, Andreas Prodromidis, and Philip Chan (1999). *KDD Cup 1999 Data*. UCI Machine Learning Repository. DOI: 10.24432/C51C7N. URL: https://archive.ics.uci.edu/dataset/130/kdd+cup+1999+data (visited on 12/13/2025).

Su, Yinghao, Dapeng Xiong, Kechang Qian, and Yu Wang (2024). "A Comprehensive Survey of Distributed Denial of Service Detection and Mitigation Technologies in Software-Defined Network". In: *Electronics* 13.4, p. 807. DOI: 10.3390/electronics13040807. URL: https://www.mdpi.com/2079-9292/13/4/807.

Sultana, S. et al. (2019). "A Survey on Intrusion Detection in Software Defined Networking". In: *Journal of Network and Computer Applications*.

Sun, Sophia Huiwen, Abishek Sankararaman, and Balakrishnan Murali Narayanaswamy (2024). "Online Adaptive Anomaly Thresholding with Confidence Sequences". In: *Proceedings of the 41st International Conference on Machine Learning (ICML)*. Vol. 235. Proceedings of Machine Learning Research, pp. 47105–47132. URL: https://proceedings.mlr.press/v235/sun24h.html.

Tang, Dan, Rui Dai, Yudong Yan, Keqiu Li, Wenjia Liang, and Zhen Qin (2024a). "FTODefender: Low-rate Flow Table Overflow Attacks Defending System in SDN". In: *Expert Systems with Applications*. DOI: 10.1016/j.eswa.2023.121460.

— (2024b). “When SDN Meets Low-rate Threats: A Survey of Attacks and Defense Strategies”. In: *ACM Computing Surveys*. DOI: 10.1145/3704434.

Tavallaee, Mahbod, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani (2009). “A Detailed Analysis of the KDD CUP 99 Data Set”. In: *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, pp. 1–6. DOI: 10.1109/CISDA.2009.5356528.

Varadharajan, Vijay and Uday Tupakula (2019). “Counteracting Attacks from Malicious End Hosts in Software Defined Networks”. In: *IEEE Transactions on Network and Service Management* 17.1, pp. 160–174.

Vasilomanolakis, Emmanouil, Shankar Karuppayah, Max Mühlhäuser, and Mathias Fischer (2015). “Taxonomy and Survey of Collaborative Intrusion Detection”. In: *ACM Computing Surveys* 47.4, 55:1–55:33. DOI: 10.1145/2716260.

Verizon DBIR Team (2024). *2024 Data Breach Investigations Report*. Tech. rep. Verizon. URL: https://www.verizon.com/business/resources/reports/2024-dbir-data-breach-investigations-report.pdf.

Wang, Feng, Donghua Zhu, Lin Li, et al. (2025). “A Survey of Deep Anomaly Detection in Multivariate Time Series”. In: *Sensors* 25.1, p. 190. DOI: 10.3390/s25010190.

Wang, Haopei, Lei Xu, and Guofei Gu (2015). “FloodGuard: A DoS Attack Prevention Extension in Software-Defined Networks”. In: *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 239–250. DOI: 10.1109/DSN.2015.27.

Wu, Yanzhao and Ling Liu (Feb. 2023). “Selecting and Composing Learning Rate Policies for Deep Neural Networks”. In: *ACM Trans. Intell. Syst. Technol.* 14.2. ISSN: 2157-6904. DOI: 10.1145/3570508. URL: https://doi.org/10.1145/3570508.

Xie, Jun et al. (2019). “Machine Learning-Based Intrusion Detection for Software-Defined Networking: A Survey”. In: *IEEE Access*.

Xin, Yu, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang (2018). “Machine Learning and Deep Learning Methods for Cybersecurity”. In: *IEEE Access* 6, pp. 35365–35381. DOI: 10.1109/ACCESS.2018.2836950.

Xu, Jiehui, Haixu Wu, Jianmin Wang, and Mingsheng Long (2022). “Anomaly Transformer: Time Series Anomaly Detection with Association Discrepancy”. In: *International Conference on Learning Representations (ICLR)*. URL: https://openreview.net/forum?id=LzQQ89U1qm_.

Yang, Limin, Zhi Chen, Chenkai Wang, Zhenning Zhang, Sushruth Booma, Phuong Cao, Constantin Adam, Alexander Withers, Zbigniew Kalbarczyk, Ravishankar K. Iyer, and Gang Wang (2024). "True Attacks, Attack Attempts, or Benign Triggers? An Empirical Measurement of Network Alerts in a Security Operations Center". In: *33rd USENIX Security Symposium (USENIX Security 24)*.

Yang, Xue, Enda Howley, and Micheal Schukat (2023). *ADT: Agent-based Dynamic Thresholding for Anomaly Detection*. arXiv: 2312.01488 [cs.LG]. URL: https://arxiv.org/abs/2312.01488.

Yoachimik, Omer and Jorge Pacheco (Jan. 2024). *DDoS threat report for 2023 Q4*. Cloudflare Blog. URL: https://blog.cloudflare.com/ddos-threat-report-for-2023-q4/.

Yoon, Changhoon et al. (2017). "Flow Wars: Systemizing the Attack Surface and Defenses in Software-Defined Networks". In: *IEEE/ACM Transactions on Networking*. DOI: 10.1109/TNET.2017.2748159.

Yu, Mingli, Quinn K. Burke, and Thomas F. La Porta (2024). "Stealthy Misreporting Attacks Against Load Balancing". In: *IEEE/ACM Transactions on Networking* 32.4, pp. 3622–3635. DOI: 10.1109/TNET.2024.3396387.

Yue, Meng, Qingxin Yan, Han Zheng, and Zhijun Wu (2022). "Cross-Plane DDoS Attack Defense Architecture Based on Flow Table Features in SDN". In: *Security and Communication Networks* 2022. DOI: 10.1155/2022/7409083.

Yungaicela-Naula, Noe M., Cesar Vargas-Rosales, Jesús Arturo Pérez-Díaz, Eduardo Jacob, and Carlos Martinez-Cagnazzo (2023). "Physical Assessment of an SDN-Based Security Framework for DDoS Attack Mitigation: Introducing the SDN-SlowRate-DDoS Dataset". In: *IEEE Access* 11, pp. 46820–46831. DOI: 10.1109/ACCESS.2023.3274577.

Zamanzadeh Darban, Zahra, Geoffrey I. Webb, Shirui Pan, Charu C. Aggarwal, and Mahsa Salehi (2024a). "Deep Learning for Time Series Anomaly Detection: A Survey". In: *ACM Computing Surveys* 57.1. DOI: 10.1145/3691338.

— (2024b). "Deep Learning for Time Series Anomaly Detection: A Survey". In: *ACM Computing Surveys* 57.1. DOI: 10.1145/3691338.

Zerbini, Cinara Brenda, Luiz Fernando Carvalho, Taufik Abrão, and Mario Lemes Proença Jr. (2019). "Wavelet Against Random Forest for Anomaly Mitigation in Software-Defined Networking". In: *Applied Soft Computing* 80, pp. 138–153. DOI: 10.1016/j.asoc.2019.02.046.

Zhang, Qi, Benjamin Turnbull, Kasra Kermanshahi, Hemanshu Pota, and Junbin Hu (2025). *SDN-MG25: An SDN-Driven Microgrid Dataset for Intrusion Detection*. Preprint (Preprints.org). DOI: 10.20944.