

A Multi-Class SDN Intrusion Detection Dataset with Synchronized OpenFlow Control-Plane Telemetry

Juliana Arévalo-Herrera ¹, Jorge E. Camargo ^{2,*}, José Ignacio Martínez Torre ¹, Juan Marcos Ramírez ³
and Tatiana Zona-Ortiz ⁴

¹ Departamento de Informática y Estadística, Universidad Rey Juan Carlos, 28933 Móstoles, Spain; j.arevaloh.2019@alumnos.urjc.es (J.A.-H.); joseignacio.martinez@urjc.es (J.I.M.T.)

² Departamento Ingeniería de Sistemas e Industrial, Universidad Nacional de Colombia, Bogotá 111321, Colombia

³ IMDEA Networks Institute, 28918 Madrid, Spain; juan.ramirez@imdea.org

⁴ Ingeniería de Telecomunicaciones, Universidad Santo Tomás, Bogotá 111321, Colombia; angelazona@usta.edu.co

* Correspondence: jecamargom@unal.edu.co

Abstract

Software-Defined Networking (SDN) separates the control and data planes, introducing a logically centralized controller that is itself a high-value attack target. Despite growing interest in SDN intrusion detection, publicly available datasets either restrict evaluation to binary normal-vs-DDoS classification or lack control-plane telemetry, leaving multi-class detection of SDN-architectural attacks without a dedicated benchmark. This work presents LAN-SDN-NIDS, a publicly available, multi-class flow-level dataset of 1,125,059 records generated in a fully containerized ContainerNet/OpenDaylight testbed across five standard network topologies. Each flow record combines 29 traffic-level features with 11 control-plane-aware metrics—including Packet-In and Flow-Mod counts and first-seen delay. The dataset covers five attack classes in two categories: three that exploit SDN control-plane mechanisms (link fabrication, host injection, and port hijack) alongside DDoS and port scan, plus normal traffic. An XGBoost classifier trained on the full feature set achieved a macro F1 of 0.94; an ablation study showed that removing OpenFlow features causes link fabrication F1 to collapse from 0.97 to 0.19, indicating that control-plane telemetry is decisive for detecting SDN-architectural attacks under the conditions evaluated. A UMAP embedding is consistent with class separability, except for a structural overlap between host injection and normal traffic attributable to their shared ARP protocol.

Dataset: <https://doi.org/10.21950/QMAXKP>

Dataset License: CC-BY-4.0



Academic Editor: Keke Chen

Received: 22 June 2026

Revised: 24 July 2026

Accepted: 30 July 2026

Published: 5 August 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: network intrusion detection; OpenFlow protocol; SDN security; control plane; ARP; LLDP; link fabrication; host injection

1. Summary

SDN decouples the control and data-forwarding planes, centralizing network intelligence in a controller that communicates with forwarding devices via the OpenFlow protocol [1]. When switches encounter unknown flows, they generate Packet-In messages to the controller—a mechanism that exposes the control plane to exploits that manipulate controller-switch interactions rather than relying on high-volume data-plane traffic.

Existing SDN-focused datasets and detection approaches present three persistent gaps (Table 1): First, DDoS detection at the data plane dominates the literature—several works train classifiers on SDN-generated datasets, achieving high binary accuracy, but restrict evaluation to normal-vs-DDoS classification, limiting attack diversity [2–5]. Second, dataset SDN-specificity is inconsistent—some studies apply models to generic datasets not designed for SDN environments, raising representativeness concerns [6], while others rely on non-public datasets, preventing reproducibility [7]. Third, multi-class detection with SDN-aware features remains absent—available multi-category benchmarks label Layer-4 service attacks [8,9], whereas attacks exploiting SDN control-plane mechanisms operate at Layer 2 through ARP and LLDP. To the best of the authors’ knowledge, the works incorporating control-plane telemetry revert to binary or coarse-grained classification [10,11].

Table 1. Comparison of public intrusion-detection datasets relevant to SDN. CP = OpenFlow control-plane telemetry used as engineered ML features (not merely captured); Sync = synchronized data-plane/control-plane collection; Abl. = OpenFlow-feature ablation reported; n/r = not reported by the source; OF-stat = OpenFlow flow-table statistics (switch counters), not control-plane event telemetry.

Dataset (Year)	Type	Controller	Topo.	Classes	Attack Focus	Flow-lvl	CP Feat.	Sync	Abl.	Size	Public
NSL-KDD (2009) [12]	Generic	–	–	5	Non-SDN	Conn.	No	No	No	148 k	Yes
CICIDS2017 (2017) [13]	Generic	–	–	8	Non-SDN	Yes	No	No	No	2.8 M	Yes
InSDN (2020) [8]	SDN	ONOS	1	8	DoS/DDoS/Probe/BFA/Web/Bot	Yes	No	No	No	343 k	Yes
Ahuja et al. (2021) [2]	SDN	n/r	1	2	DDoS (data plane)	Yes	No	No	No	104 k	Yes
Ahuja et al. (2022) [10]	SDN	Ryu	1	3	ARP poison/flood (control plane)	Yes	OF-stat	No	No	134 k	Yes
SDNFlow (2024) [14]	SDN	n/r	n/r	3	DDoS, Port scan	Yes	OF-stat	No	No	663 k	Yes
HLD-DDoSDN (2024) [4]	SDN	POX	1	4	DDoS (ICMP/TCP/UDP, controller)	Yes	No	No	No	n/r	Yes
Karmous et al. (2025) [5]	SDN-IoT	Ryu	5	2	DDoS (multi-protocol)	Yes	No	No	No	4.64 M	Yes
CSDN (2026) [15]	SDN	n/r	n/r	12	DoS/DDoS/Scan/MITM ...	Yes	n/r	n/r	No	1.59 M	Yes
ASEADOS (2026) [9]	SDN-IoT	ONOS	1	5	DoS/DDoS/Probe/Botnet	Yes	No	Yes	No	457 k	Yes
This work	SDN	ODL	5	6	OF control-plane/L2 (link fabrication, MAC-spoof, inj.)	Yes	Yes (11)	Yes	Yes	1.13 M	Yes

Table 1 compares the datasets along dimensions that can be verified from public sources. Several further criteria, however, are reported too inconsistently to tabulate, and we summarize them here.

Regarding the origin of the traffic, most public SDN datasets are generated in emulated testbeds built on Mininet or Containernet with Open vSwitch [4,5,8]. SDNFlow is the main exception, as its authors report statistics gathered from real traffic [14]. ASEADOS occupies an intermediate position, since it combines physical IoT devices with virtual nodes under a single ONOS controller [9]. The proposed dataset follows the emulated approach but extends it across five distinct topologies rather than a single fixed one.

Additionally, looking at the scale, the reporting is incomplete. Host and switch counts are stated for only a minority of the surveyed datasets, so a fair tabulation is not possible. For example, ASEADOS uses a single OVS instance with four emulated hosts and five virtual sensors [9], while HLD-DDoSDN [4] uses a single switch. Our testbed instead varies the number of hosts and switches across its five topologies, and the exact per-topology counts are reported in Section 3.1.

Class balance also differs sharply across the datasets. CSDN illustrates the common problem, as its subclasses range from a few dozen samples to more than 160,000. Such extreme skew makes adequate representation of those classes difficult. The proposed dataset is not perfectly balanced either, yet its six classes remain within a workable range, with at least a few thousand samples, and the full distribution is given in Section 2.

On the other hand, labeling is usually performed per scenario; that is, each attack is run in isolation and every resulting flow inherits the corresponding label [5,9]. Reproducibility artifacts vary more. While two datasets [14,15] require a subscription for full access, the collection presented in Table 1 is mostly public. However, only ASEADOS and the proposed work also release the complete testbed framework, whereas most others distribute the processed CSV alone.

This comparison shows that multi-class detection and OpenFlow-derived features are not, in isolation, absent from the literature, but the specific combination remains unaddressed. This work fills that gap by providing a multi-class SDN-specific dataset with Layer-2 and Layer-4 attacks with synchronized OpenFlow control-plane telemetry. It was designed with a diversity of topologies and protocols to provide conditions that better reflect operational networks.

In particular, the dataset provides labeled Layer-2 bidirectional flow records for Software-Defined Networking (SDN) security research, covering benign traffic and five SDN-relevant attack classes: link fabrication, host injection, port hijacking, DDoS, and port scan. The inclusion of synchronized OpenFlow control-plane events enables analysis of controller behavior, flow-rule dynamics, and longitudinal changes in flow-table activity during anomalous conditions.

The traffic was generated in a Containernet [16] environment controlled by OpenDaylight (ODL). A custom Python/Scapy multithreaded script simulated LAN behaviors (HTTP, SSH, DNS, FTP, RTP) consistent with SDN deployment, with stochastic inter-arrival times. Data was captured via tshark at both switch and controller interfaces. Using a custom aggregation algorithm, packets were grouped into bidirectional Layer-2 flows (30 s timeout for normal; 5 s for ARP/LLDP). These were correlated with OpenFlow telemetry (Packet-In/Flow-Mod) over each flow's active interval, extended by 10 ms for sub-millisecond flows (Section 3.3).

This dataset underlies an ongoing independent research project investigating LLM-agent-orchestrated frameworks for automated SDN attack detection and response. Preliminary validation using gradient-boosted classifiers achieved a macro-averaged F1-score of 0.94 across six classes, with the ablation study indicating that link fabrication detection depends on the OpenFlow control-plane features.

2. Data Description

The dataset contains 1,125,059 records in tabular form, released as seven files: five per-topology CSV files corresponding to five separate experiments on the same SDN testbed, plus consolidated CSV and Parquet versions (Table 2). Each record represents a Layer-2 bidirectional flow, defined as the set of packets sharing the same last-layer protocol and exchanged between a pair of hosts identified by their MAC addresses. The flow timeout varies by protocol: 30 s for general traffic and 5 s for ARP and LLDP. The shorter value matches the single-frame, millisecond-scale nature of these exchanges, in which a 30 s timeout would merge into artificial multi-exchange flows. Layer-2 flows are particularly relevant in SDN environments, where data-link layer protocols such as ARP and LLDP constitute a substantial share of control-plane traffic, as presented in Figure 1. Each record contains 40 features and one label assigned to one of six classes: normal traffic, or one of five attack types crafted specifically to target the SDN controller and its southbound interface.

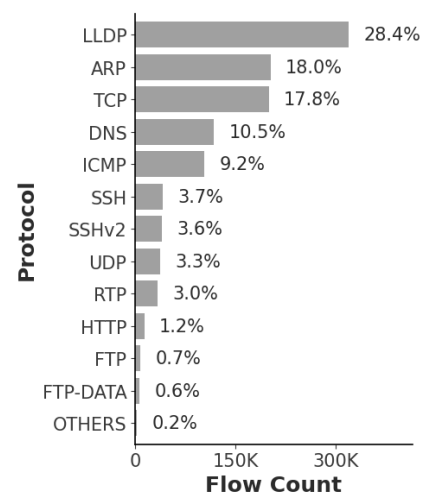


Figure 1. Protocol distribution across the five experiments.

Table 2. Dataset file summary.

	File (Topology)	Size (MB)	Records	Normal	Host Injection	Link Fabrication	Port Hijack	DDoS	Port Scan
1	linear.csv	85.3	236,401	226,809	2817	1960	2001	1629	1185
2	tree.csv	82.8	230,592	223,155	2516	196	2006	1612	1107
3	star.csv	70.7	191,664	184,260	2370	2154	1223	999	658
4	mesh.csv	83.5	232,600	225,402	2264	2170	1169	955	640
5	subnets.csv	82.1	233,802	224,105	2759	2155	2004	1632	1147
6	FULL_SDN_NIDS.csv	326.6	1,125,059	1,083,731	12,726	8635	8403	6827	4737
7	FULL_SDN_NIDS.parquet	79.3	1,125,059	1,083,731	12,726	8635	8403	6827	4737

Table 3 presents the 40 features extracted for each flow record, organized into four functional groups. The first group covers flow identification fields, including the highest observed protocol layer (last_layer), Ethernet type (eth_type), ARP operation code (arp_opcode), and the source topology (topology).

Table 3. Feature descriptions of Layer-2 flow records.

Feature Name	Description
Identification	
last_layer	Highest OSI-layer protocol observed in the flow (e.g., HTTP, SSH, ARP, LLDP)
eth_type	Ethernet type field value identifying the encapsulated protocol (e.g., 0x0800 = IPv4, 0x0806 = ARP)
arp_opcode	ARP operation code (1 = request, 2 = reply); null for non-ARP flows
topology	Source experiment identifier corresponding to the network topology used (linear, tree, star, mesh, subnet)
Volume and size	
packets	Total number of packets exchanged in the flow (forward + backward)
fw_packets	Number of packets transmitted in the forward direction
bk_packets	Number of packets transmitted in the backward direction
bytes_total	Total number of bytes exchanged in the flow (forward + backward)
fw_bytes	Total bytes sent in the forward direction
bk_bytes	Total bytes sent in the backward direction
packet_size_max	Maximum packet size observed in the flow (bytes)

Table 3. Cont.

Feature Name	Description
packet_size_min	Minimum packet size observed in the flow (bytes)
packet_size_range	Difference between maximum and minimum packet sizes; measures payload variability
fw_bytes_mean	Mean size of packets in the forward direction (bytes)
fw_bytes_median	Median size of packets in the forward direction (bytes)
fw_bytes_stddev	Standard deviation of packet sizes in the forward direction
bk_bytes_mean	Mean size of packets in the backward direction (bytes)
bk_bytes_median	Median size of packets in the backward direction (bytes)
bk_bytes_stddev	Standard deviation of packet sizes in the backward direction
duration	Total elapsed time of the flow from first to last packet (seconds)
active_time	Cumulative time during which packets were actively transmitted within the flow
idle_time	Cumulative idle time within the flow (gaps between bursts)
Temporal behavior	
interarrival_time_mean	Mean inter-packet arrival time across all packets in the flow (seconds)
interarrival_time_stddev	Standard deviation of inter-packet arrival times; captures burstiness
packets_per_second	Average packet transmission rate (packets/duration)
bytes_per_second	Average byte transmission rate (bytes_total/duration)
flow_reversals	Number of direction changes within the flow; indicates bidirectional interactivity
forward_to_backward_packet_ratio	Ratio of forward to backward packet count; values far from 1 indicate asymmetric flows
forward_to_backward_byte_ratio	Ratio of forward to backward byte count; captures upload/download asymmetry
Control-plane awareness	
OFqty	Total number of OpenFlow messages correlated with the flow during its lifetime
OFqty_packetin	Number of OpenFlow Packet-In messages associated with the flow
OF_flow_mod	Number of OpenFlow Flow-Mod messages (flow rule installations) correlated with the flow
OFpredominant_type	Most frequent OpenFlow message type observed during the flow lifetime
OFpredominant_qty	Count of the predominant OpenFlow message type
OF_activity_per_s	Rate of OpenFlow messages per second during the flow duration
OF_first_seen_delay	Time elapsed between the first data-plane packet and the first correlated OpenFlow event (seconds)
OF_to_packet_ratio	Ratio of total OpenFlow messages to total packets; high values indicate control-plane overhead
OF_to_byte_ratio	Ratio of total OpenFlow messages to total bytes exchanged
OF_packetin_ratio	Fraction of OpenFlow messages that are Packet-In events (OFqty_packetin/OFqty)
new_mac_ratio	Proportion of flows in the current 60 s window whose source MAC was absent from the preceding window

The second captures volume and size statistics in both forward and backward directions, providing asymmetry signals relevant to scanning and host injection attacks. The third group encodes temporal behavior through inter-arrival times, active and idle periods, and transmission rates, which are sensitive to low-rate control-plane manipulations. The fourth group contains eleven OpenFlow and control-plane-aware features derived by correlating data-plane flows with telemetry over each flow's active interval (Section 3.3); these include Packet-In and Flow-Mod counts, first-seen delay, and control-to-data ratios that are not present in any comparable public SDN dataset.

Among the traffic-derived features, `new_mac_ratio` warrants a precise definition. It is computed over a sliding 60 s window rather than the full capture. For a flow at time t , let $F_t = \{i : t_i \in [t - 60\text{ s}, t]\}$ be the flows in the current window, m_i the source MAC of

flow i , and $M_t^- = \{m_j : t_j \in [t - 120\text{ s}, t - 60\text{ s}]\}$ the set of source MACs observed in the preceding window. Then,

$$\text{new_mac_ratio} = \frac{|\{i \in F_t : m_i \notin M_t^-\}|}{|F_t|}, \quad (1)$$

and 0 when F_t is empty. The value ranges from 0, when every address was present in the prior window, to 1, when all are newly appearing. Because the reference is a rolling prior window rather than the full history, the feature measures a short-term novelty rate and does not reduce to an inverse packet count. It stores no address value, only an aggregate rate, so no host identity is recoverable from it. Although computed from data-plane headers, it is grouped with the control-plane features because an unseen source MAC produces a flow-table miss and, hence, a Packet-In and a host-tracking update at the controller.

The remaining ten features in this group derive directly from correlated OpenFlow events. All message types are counted, and `OFpredominant_type` is categorical, ordinally encoded before training. Ratios whose denominator is zero—for example `OF_packetin_ratio` when `OFqty` equals 0—are set to 0. A machine-readable data dictionary specifying the data type, unit, and value range of each feature is provided as `data_dictionary.csv` in the dataset repository. It keys on the source MAC and does not reference the reserved destination-MAC labeling prefixes (Section 3.4).

While the feature count is lower than general-purpose SDN datasets such as InSDN [8], 11 of the 40 features are control-plane-aware. Among the datasets surveyed in Table 1, control-plane features, where present at all, are limited to OpenFlow flow-table statistics; engineered control-plane event features remain unaddressed, providing signals specific to SDN-architectural exploits.

Five distinct network topologies were implemented in Containernet: linear, tree, star, mesh, and subnet-based configurations. For each topology, benign traffic was generated using scripted background generators, while attack traffic was induced deterministically via MAC address spoofing specific to each attack class. Attacker hosts, iteration counts, and timing were randomized per run. Table 2 summarizes the file breakdown and class distribution per topology. There is one CSV file per experiment, each one with a different topology. Two additional consolidated files are provided: one CSV (`FULL_SDN_NIDS.csv`) and one parquet (`FULL_SDN_NIDS.parquet`), merging all five topologies into a single cleaned version suitable for direct use in machine learning pipelines with a total of 1,125,059 records.

The dataset encompasses the protocols typical of LAN environments. As shown in Figure 1, LLDP (28.4%) and ARP (18%) dominate the flow distribution, which is expected given their central role in SDN topology discovery and host resolution at the data-link layer. TCP (17.8%), DNS (10.5%), and ICMP (9.2%) constitute the bulk of the remaining traffic, complemented by SSH, SSHv2, UDP, RTP, HTTP, FTP, and FTP-DATA. A small number of flows (2255, <0.2% of the dataset) carry protocol identifiers associated with legacy services on well-known ports (e.g., ECHO port 7, DAYTIME port 13, Chargen port 19), produced during port scan execution; these records are retained in the consolidated dataset as their labels are correctly assigned by a MAC-based labeling mechanism, described in Section 3.4.

Each experiment comprises six classes, with “normal” the most abundant (96.3%). Total runtime is not fixed in advance: attacks run for randomized iterations and cooldown gaps until the target flow count per class—which is a configurable parameter—is reached, so runtime emerges from these durations and the stochastic traffic intensity, yielding capture spans of 292.0, 292.0, 292.8, 292.7, and 293.1 min for linear, tree, star, mesh, and subnets, respectively. The five attack classes target different parts of the SDN architecture. Each record is labeled with one of six classes: normal traffic, or one of five attacks—link

fabrication, host injection, port hijack, DDoS, and port scan. The attack mechanisms and taxonomy are described in Section 3.2.

Figure 2 shows the specific target of each attack. Link fabrication poisons the topology database, host injection corrupts host tracking, and port hijack overwrites host-location binding; all three originate in the data plane but target the control-plane state. DDoS exhausts controller CPU resources, and port scan probes forwarding behavior at the data plane.

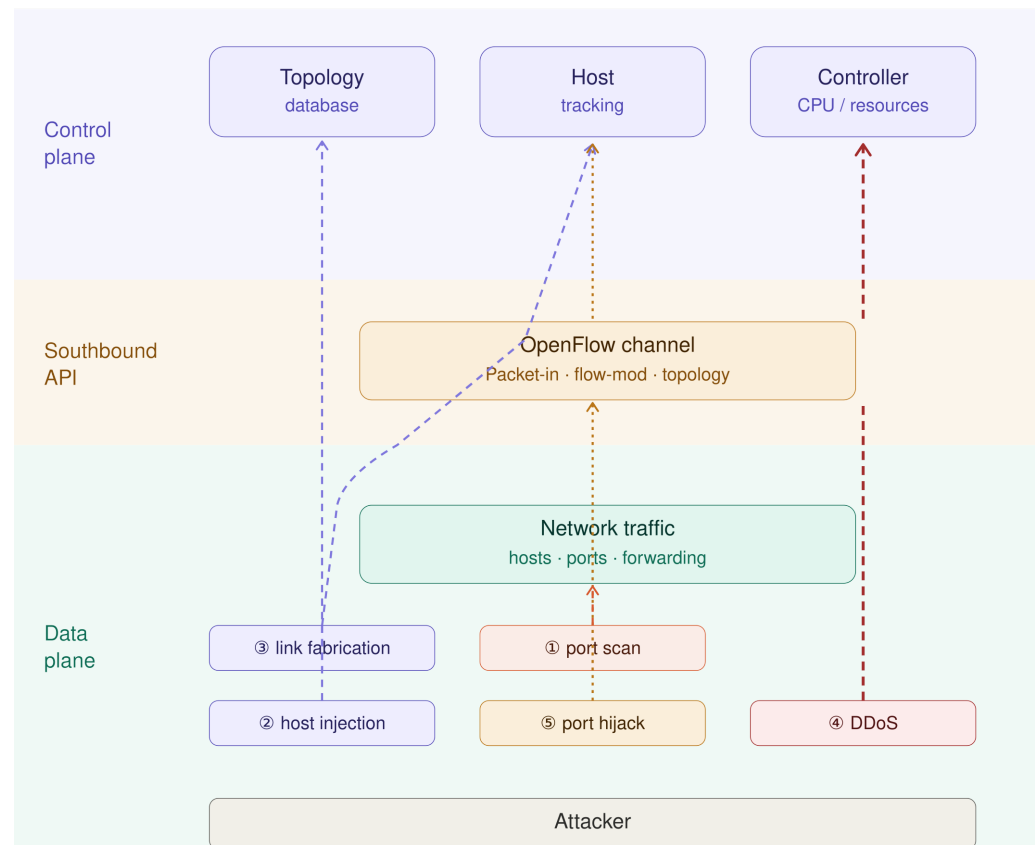


Figure 2. SDN architectural components targeted by each attack class.

Within the dataset, the distribution of the classes is presented in Figure 3. Benign traffic dominates the capture, while attacks occur as intermittent, low-volume events. The attack records (host injection, link fabrication, port hijack, DDoS, and port scan) represent less than 5% of the total, yielding a pronounced class imbalance typical of real networks. The class distribution is intentionally imbalanced, since a skewed distribution provides a more demanding benchmark than a balanced one, requiring classifiers to maintain per-class performance on minority attack classes.

Researchers should account for this when designing classifiers; accuracy is not a suitable primary metric for this dataset. Instead, metrics robust to imbalance—such as macro-averaged F1-score or Area Under the Precision–Recall Curve (AUPRC)—are recommended. Resampling strategies such as SMOTE or cost-sensitive learning may also improve minority class detection.

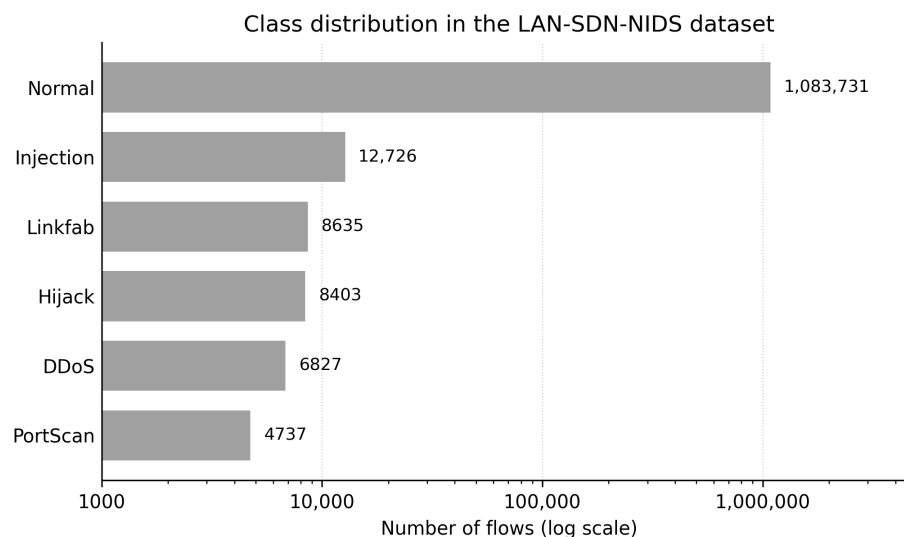


Figure 3. Class distribution in the dataset.

3. Methods

The dataset was generated using a containerized testbed, where a controller manages multiple OpenFlow switches connecting corporate hosts and services, while one or more compromised internal nodes launch SDN-specific attacks.

The infrastructure was deployed on an Intel(R) Core(TM) i7-7700 CPU @ 3.60 GHz with 32 GB of memory, running Debian GNU/Linux 12 (bookworm) with kernel 6.1.0-13-amd64. The containerized environment is supported by Docker Engine for isolation and reproducibility. The control plane runs an OpenDaylight (ODL) controller managing the network's global state. Exact software versions, image identifiers, and package dependencies are provided in the `environment.md` file accompanying the dataset. The data plane is deployed with an Orchestrator script that coordinates Python-based topology definitions and traffic generation. The host containers execute a multithreaded Python script managing three concurrent services: traf-gen for stochastic baseline traffic (HTTP, SSH, DNS, FTP, RTP), attacks for SDN-specific exploit injection, and servers to provide legitimate response targets. The architecture is presented in Figure 4.

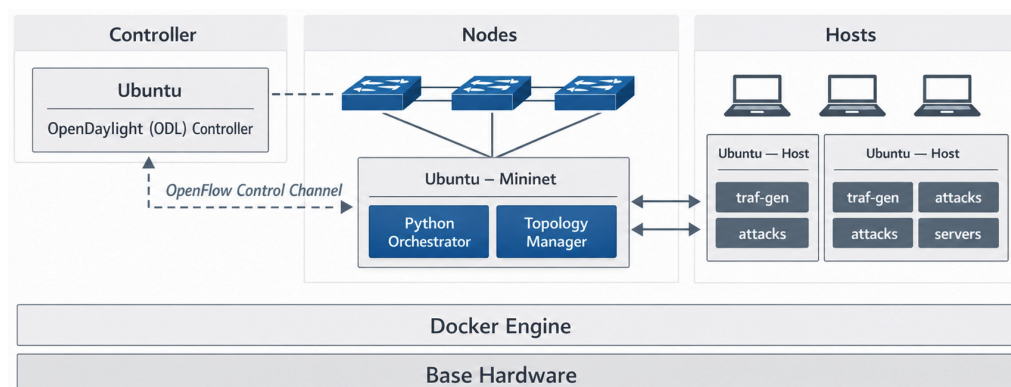


Figure 4. Containerized SDN testbed architecture.

3.1. Orchestrator and Topology Management

An orchestrator script manages experiment execution by forking two concurrent threads after the setup phase: an attack thread and a traffic thread. Configurable parameters include the network topology (five distinct options described below), the target flow count per attack, and randomized timing elements applied to iteration counts and gap

durations to avoid deterministic traffic patterns. Figure 5 shows the model timing for a single execution. The traffic thread simulates three behavioral phases running concurrently with the attack thread. The attack thread executes five attacks sequentially, separated by 10-s cooldown gaps. The orchestrator initiates packet captures during the setup phase on both the flow-capture hosts attached to each switch and the OpenDaylight controller.

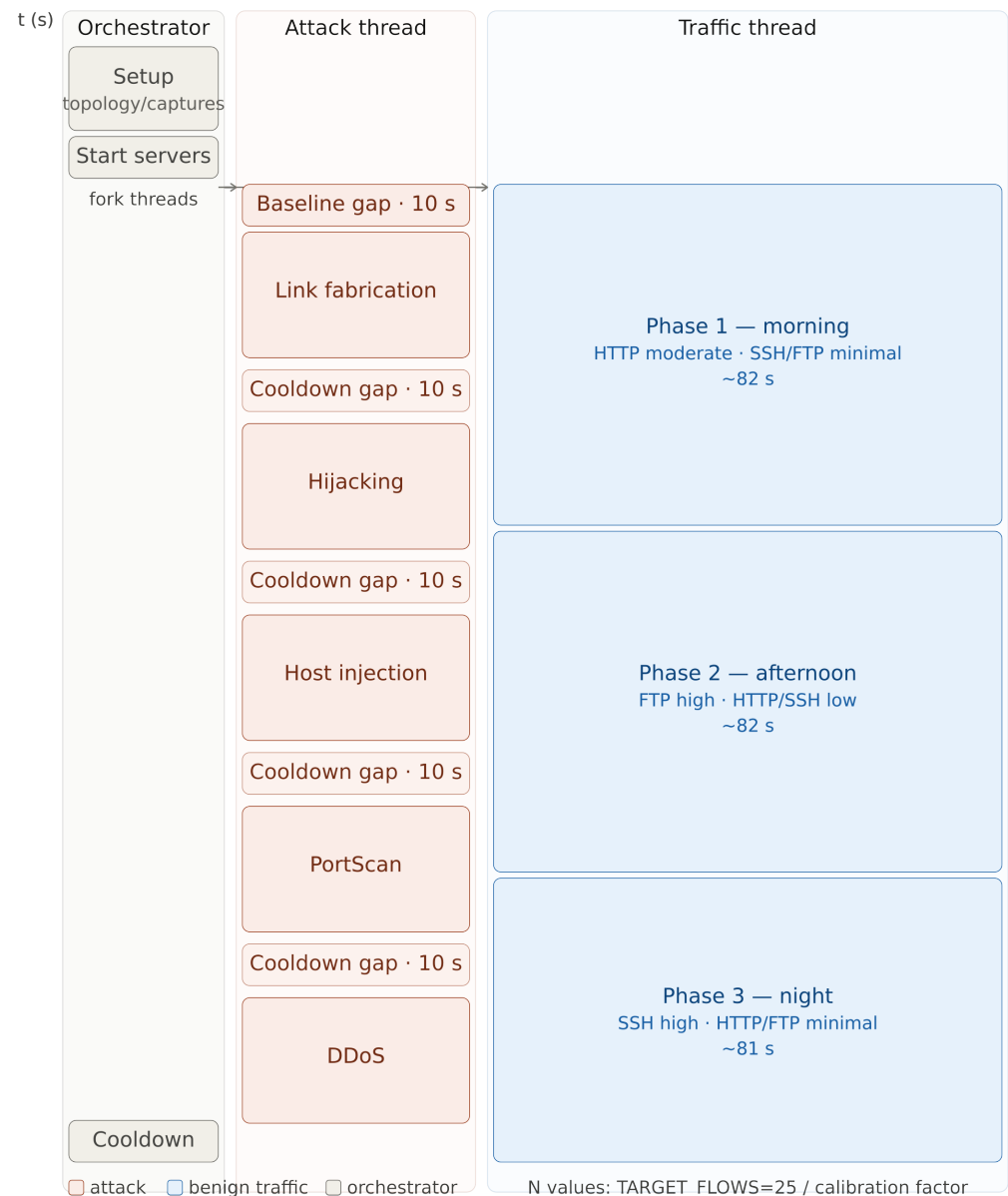


Figure 5. Execution timeline per experiment.

A total of five different topology managers were created, one for each script, using the Mininet Python API used by Containernet. For each switch, several flow-capture hosts and mirroring ports were created with the intention of capturing all traffic from the switches to carry out a later process to create the dataset. Table 4 shows the configuration of each experiment.

Regarding the benign traffic, a Python script (version 3.11.2) was created to emulate a normal user. Table 5 presents the protocols and behavior. Together with benign control traffic (LLDP and ARP), this accounts for the 96.3% of flows labeled normal; the remainder corresponds to attacks.

Table 4. Testbed configuration per experiment.

Experiment	Network Topology	Switches	Traffic Hosts	Flow-Capture Hosts
1	Linear	6	19	6
2	Tree	7	21	7
3	Star	5	11	5
4	Mesh	4	12	4
5	Mixed/Server Subnet	6	18	6

Table 5. Benign traffic generation modules and simulated behaviors.

Module	App Layer	Transport	Port	Simulated Behavior	Library
HTTP	HTTP/1.1	TCP	80	GET requests to randomized paths (/ , /login, /api/status, ...); rotating User-Agent strings; occasional DNS noise injection	requests
SSH	SSHv2	TCP	22	Persistent sessions (40–120 s); randomized shell commands (whoami, ls -la, uname -r, ...)	paramiko
FTP	FTP	TCP	21	Anonymous login; LIST directory; periodic N00P keepalives	ftplib
ICMP	—	ICMP	—	Echo Request/Reply (sr1); 32-byte payload mimicking default OS ping; waits for reply to capture RTT	Scapy
DNS	DNS	UDP	53	Recursive A-record queries for a pool of real domains; sr1 waits for resolver response	Scapy
RTP	RTP	UDP	5004	G.711 PCMU simulation: 160-byte payload, 20 ms packet interval; call sessions of 30–180 s; monotonic seq/timestamp counters	Scapy

Benign traffic generation is organized into three consecutive behavioral phases, each running for a third of the calculated time for attacks, as designed assumptions, not derived from empirical traces; the generators reproduce protocol structure rather than the statistical richness of real user behavior. The morning phase simulates moderate HTTP activity with minimal SSH and FTP usage; the afternoon phase shifts to high-volume FTP transfers with reduced HTTP and SSH; and the night phase emphasizes SSH sessions with minimal HTTP and FTP. Attack execution and benign traffic generation run concurrently from the moment the orchestrator forks both threads, ensuring that attack flows are embedded within background traffic rather than isolated in an otherwise idle network.

The hosts also have access to the attack scripts, but only a randomly selected set execute them, according to the orchestrator. Each type of attack is tagged using forged MAC addresses with Scapy, so they can be used later to label the flow in the appropriate manner. Section 3.2 describes the mechanisms used for each attack.

3.2. Attack Descriptions

We distinguish two attack categories: Link fabrication, host injection, and port hijack are SDN-architectural attacks; they exploit control-plane mechanisms (topology discovery, host tracking, and host-location binding) directly. DDoS and port scan are conventional attacks whose effect is observed through the control plane, via Packet-In saturation and reconnaissance activity, respectively. Attacks are classified by the mechanism they target, not by the plane from which the packets originate. Table 6 presents an overview of each one, and the section below presents detailed descriptions for each one.

Table 6. Attack descriptions, targets, and expected effects.

Attack	Targeted Component	Mechanism	Expected Effect	Dataset Label
Link fabrication	Control plane—Topology discovery	Forged LLDP packets from two malicious hosts to make the controller register a non-existent link between two switches	Corrupted topology view; potential for traffic misdirection	Linkfab
Host injection	Control plane—Host tracking	Packets with spoofed source MAC/IP cause the controller to register non-existent hosts in its host database	Polluted host inventory; misconfigured forwarding rules	Injection
Port hijack	Control plane—Host location	Forged ARP replies bind the target host's IP to the attacker's MAC, overwriting its location at the controller	Identity spoofing; traffic interception or blackholing	Hijack
DDoS	Control plane—Controller load	High-rate flood with randomized source addresses generates a continuous stream of Packet-In requests to the controller	Controller CPU saturation; degraded network-wide responsiveness	DDoS
Port scan	Data plane—Reconnaissance	Randomized TCP SYN/UDP probes across ports 1–1024, with inter-packet delay drawn from $\mathcal{U}(0.001, 0.1)$ s	Network exposure; early indicator of a multi-stage attack	Port scan

3.2.1. Host Injection Attack

A compromised host executes a script that crafts ARP packets using the Scapy Python library. These fraudulent packets are injected into the network and forwarded to the controller via Packet-In OpenFlow messages. Over time, this behavior leads the controller to perceive the existence of non-existing hosts.

Once at least one host is compromised, an attacker can launch a simple flow of ARP packets that forces the switch to repeatedly request instructions from the controller using the Packet-In mechanism.

The compromised host uses a script to craft ARP packets. Following the method proposed in [17], the poisoning attack follows the structure presented in Algorithm 1.

Algorithm 1 Non-existing host attack algorithm

```

 $r \leftarrow \text{RandomMAC}()$ 
 $\text{NewPacket} \leftarrow \text{ARP}(\text{src} = r)$ 
 $\text{limit} \leftarrow \text{SizeOfFlow}$ 
 $\text{cont} \leftarrow 0$ 
while  $\text{cont} \leq \text{limit}$  do
     $\text{NetworkInterface} \leftarrow \text{NewPacket}$ 
     $\text{cont} \leftarrow \text{cont} + 1$ 
end while

```

The crafted flow forces switches to request information from the controller to create new entries in the flow table by issuing Packet-In OpenFlow messages. In this scenario, the controller must identify the packet origin, destination, and traffic type in order to define the appropriate forwarding action, which is then sent to the switches. As a result, the controller updates its internal records to include the new hosts sending ARP packets, even though these hosts are fictitious.

When the attacker executes the script on the compromised host, the controller begins receiving Packet-In OpenFlow messages through the control channel interface. A packet sniffer attached to this interface reveals that the payload of these messages consists of ARP requests that the switch cannot process locally due to the absence of matching flow-table entries.

The controller interprets these Packet-In messages as indicators of topology changes and updates its records accordingly. The graphical console of OpenDaylight enables visualization of the topology as perceived by the controller. Figures 6 and 7 show the topology before and after the attack.

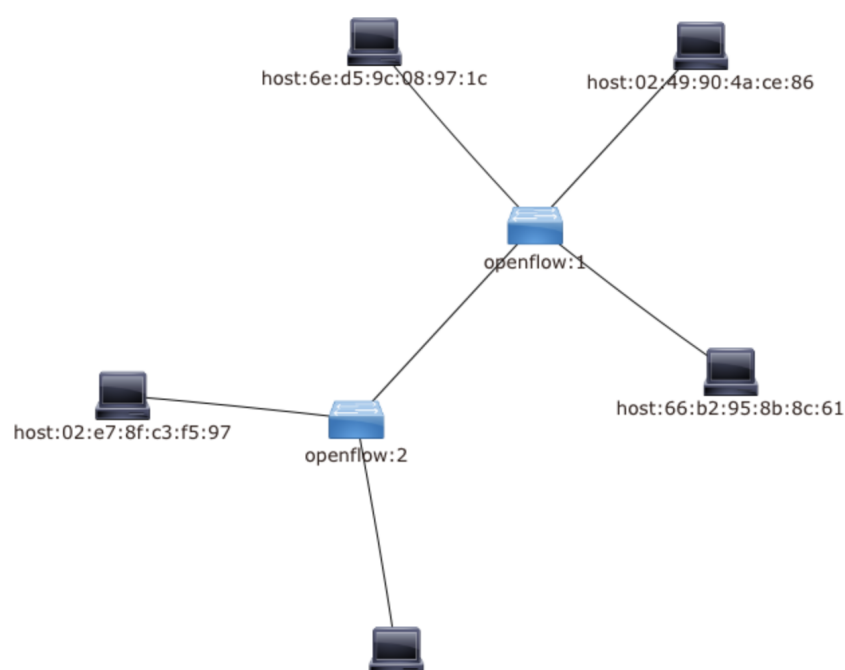


Figure 6. Topology in ODL prior to the attack.

3.2.2. Port Hijacking

Port hijacking injects forged ARP packets (eth_type 0x0806) into the network. The attacker spoofs the hwsrc and psrc fields, advertising the target host's IP address bound to the attacker's own MAC address. Consequently, the controller re-associates the legitimate host's identity with a switch port under the attacker's control. The forged frame is delivered as a rapid burst of ARP packets to the broadcast address (ff:ff:ff:ff:ff:ff). It is represented as

$$\mathcal{F} = \{ \text{Eth}(dst = BC, src = MAC_{atk}) \parallel \text{ARP}(hwsrc = MAC_{atk}, psrc = IP_{tgt}) \} \quad (2)$$

where MAC_{atk} is the attacker's hardware address and IP_{tgt} is the target host's IP address. The switch forwards these packets to the controller via Packet-In messages. The controller then updates its host-tracking database, rebinding the target host's location to the attacker's port. As a result, traffic destined for the target host is redirected toward the attacker.

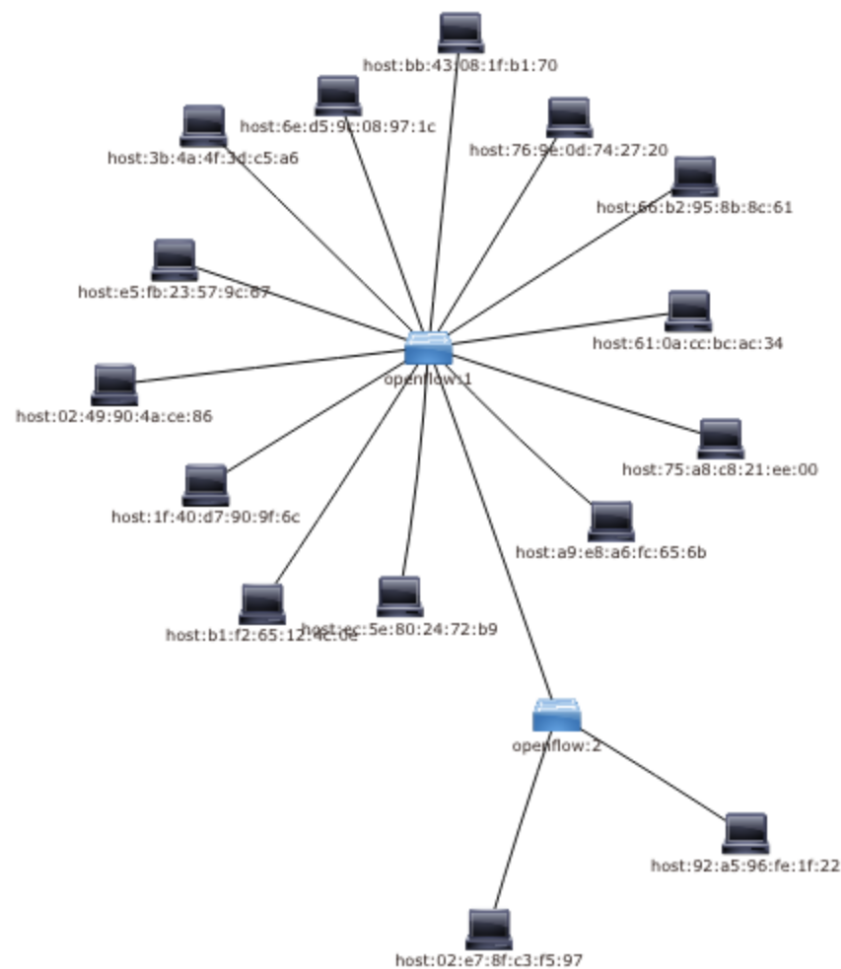


Figure 7. Topology in ODL after the attack.

3.2.3. Link Fabrication Attack

The link fabrication attack is a topology poisoning vector that exploits the trust relationship between the SDN controller and the Link Layer Discovery Protocol (LLDP).

The attack requires two compromised hosts equipped with an out-of-band communication channel to exchange intercepted LLDP frames that will create a fake link between the two switches to which they are connected. The attack follows a two-stage methodology:

1. **LLDP Interception and Exfiltration:** Using the `sniff` function and a custom `to_dataframe` parser, the compromised host (*victim_a*) captures legitimate LLDP packets (*EtherType* : 0x88cc) transmitted by the switch. These packets contain critical TLV (Type–Length–Value) fields, including ChassisID and PortID.
2. **Frame Reconstruction and Injection:** The extracted metadata is shared with the second compromised host (*victim_b*), which uses Scapy to reconstruct an identical LLDP frame. By injecting this cloned frame into its local switch port, the attacker deceives the OpenDaylight (ODL) controller into believing that a direct physical link exists between the switch ports associated with *victim_a* and *victim_b*.

As described in [18], this attack creates a “ghost link” in the SDN controller’s internal topology graph. The logic of the attack is presented in Algorithm 2.

Algorithm 2 LLDP attack algorithm

```

1: while True do
2:   sniffedPacket  $\leftarrow$  NetworkInterfaceLldpPacket
3:   LldpPacketFileOut  $\leftarrow$  SniffedPacket
4:   CompromisedChannel  $\leftarrow$  LldpPacketFileOut
5:   LldpPacketFileIn  $\leftarrow$  CompromisedChannelPacket
6:   NewPacket  $\leftarrow$  LLDP(LldpPacketFileIn)
7:   NetworkInterface  $\leftarrow$  NewPacket
8: end while

```

3.2.4. Port Scan Attack

The port scan attack in the SDN testbed simulates a reconnaissance phase in which an internally compromised host attempts to enumerate the services running on the designated *web_host*. The attack uses the Scapy library to craft TCP SYN and UDP probe packets targeting ports in the range 1–1024. This behavior exhibits distinct characteristics in an SDN environment compared to traditional network architectures.

To prevent the dataset from exhibiting the deterministic or “robotic” signatures commonly observed in legacy datasets such as KDD-99, stochastic behavior is introduced into the attack manager. Specifically, the inter-packet delay (t_{inter}) is randomized within the interval $[0.001, 0.1]$ seconds, as defined by

$$t_{inter} \sim \mathcal{U}(0.001, 0.1). \quad (3)$$

This variability forces machine learning models to detect anomalies based on flow dynamics rather than simple fixed-rate thresholds.

3.2.5. DDoS Attack

For DDoS, three randomly selected compromised hosts execute the same Scapy-based script concurrently, parameterized by inter-packet delay, packet quantity per burst, attack duration, a host identifier, and the transport protocol. The orchestrator assigns each run a protocol chosen at random (70% TCP, 30% UDP). The script dynamically selects the active network interface and continuously injects packets toward randomly generated IP addresses within the 10.0.1.0/24 subnet: TCP runs send SYN packets to port 80, whereas UDP runs target random destination ports. Each packet carries a 512-byte random payload. The destination MAC address is forged using a deterministic pattern derived from the host identifier, enabling controlled differentiation among attackers and labeling.

Each script stops five seconds before its nominal duration, leaving a short quiet window for the controller to stabilize. During the attack window, each host repeatedly sends bursts of packets, defined by the packet quantity parameter, with a configurable inter-packet interval. This behavior results in a sustained stream of new, previously unseen flows.

From the SDN perspective, the switches are unable to match these packets against existing flow-table entries and, therefore, forward them to the controller using Packet-In OpenFlow messages. As multiple compromised hosts execute the script concurrently, the controller experiences a sharp increase in Packet-In events, forcing it to repeatedly process flow setup decisions. This leads to elevated control-plane load, increased flow-table churn, and delayed response times for legitimate traffic.

Unlike volumetric DDoS attacks in traditional networks, this attack exploits the reactive nature of SDN forwarding. Its effectiveness stems not only from traffic volume but from the continuous creation of short-lived flows that trigger controller involvement. As a result, the attack is particularly suitable for evaluating ML/DL models designed to detect distributed, SDN-specific DDoS behavior at the flow level.

3.3. Plane Synchronization

Data-plane and control-plane captures were taken on the same host under a single system clock, so both share one time source, and no cross-machine clock synchronization is required. Each Layer-2 flow is correlated with the OpenFlow events whose timestamps fall within the flow's active interval $[t_{\text{start}}, t_{\text{end}}]$, where t_{start} and t_{end} are the flow's first and last data-plane packets. All events in the interval are counted, not matched one-to-one: `OFqty` is their total, `OFqty_packetin` and `OF_flow_mod` their per-type counts, and `OFpredominant_type` the most frequent type. A flow therefore admits zero, one, or several events without ambiguity, since events are aggregated per flow rather than assigned uniquely.

Flows shorter than 1 ms—predominantly single-frame ARP and LLDP exchanges—would yield a near-zero-width interval that captures no events, even when the switch did raise a Packet-In for that frame. For these flows, the interval end is padded by 10 ms. This value was chosen empirically: without the pad, the OpenFlow features were uniformly zero for ARP and LLDP flows, erasing the control-plane signal for precisely the classes (host injection, link fabrication) that depend on it. In this single-host testbed, where control traffic traverses intra-host virtual switching, 10 ms is a conservative upper bound on controller reaction latency, so a wider pad would not add events for these flows. The pad is a fixed generation parameter; the flow-building and correlation scripts are released so that users requiring a different value can regenerate the features.

For flows of 1 ms or longer the window is the flow duration. The 10 ms pad applies only to A% of flows, where the observed `OF_first_seen_delay` stays below it. No window sweep was performed, so this parameter conditions the OpenFlow results.

Each flow is correlated over its own duration, so the only free parameter is the 10 ms pad. It applies only to sub-millisecond ARP and LLDP flows. A smaller pad zeroes their OpenFlow features. A larger one adds nothing, since 10 ms exceeds the controller reaction latency in this testbed. No sensitivity sweep was run. The reported control-plane contribution to link fabrication and host injection is therefore conditional on this value.

3.4. Ground-Truth Labeling

Labels are assigned by a deterministic offline oracle rather than inferred from traffic behavior. During generation, each attack script crafts packets whose destination MAC address belongs to a reserved, per-attack address block that is never assigned to a legitimate host. Table 7 lists the five reserved prefixes and their corresponding classes.

Table 7. Reserved destination-MAC prefixes used for ground-truth labeling.

Destination-MAC Prefix	Assigned Class
aa:aa:aa:aa:aa:	Link fabrication
aa:aa:aa:aa:bb:	Port hijack
aa:aa:aa:aa:cc:	Host injection
aa:aa:aa:aa:dd:	DDoS
aa:aa:aa:aa:ab:	Port scan
(no match)	Normal

Then, after capture and flow construction, each flow is matched against this fixed table by its destination-MAC prefix. Consequently, flows carrying a reserved prefix receive the corresponding attack label, while all remaining flows are labeled normal. Because each attack uses a dedicated block disjoint from the legitimate address space, the mapping from prefix to class is unambiguous and requires no inference from flow behavior.

The identifiers used for labeling are not present in the features that the classifier observes. Raw source and destination MAC and IP addresses are removed in a later step,

when the individual flow files are consolidated into the unified training file (Section 3.5). Therefore, the reserved prefixes act purely as an external ground-truth oracle: the labeling field is dropped before training, and no traffic-level feature is a copy or a reversible encoding of it. As a result, label assignment and feature representation are disjoint by construction, which avoids circularity between the labeling mechanism and the traffic-derived features.

A leakage-specific ablation is left to future work; the control-plane block, which contains `new_mac_ratio`, alone yields macro F1 0.60 (Table 8), inconsistent with a feature encoding the label.

Table 8. Feature ablation analysis for multi-class attack detection (F1-score, precision, and recall).

Class/Metric	Precision	Recall	F1-Score	Support
Full (40 features)				
DDoS	1.00	1.00	1.00	1366
Port hijack	1.00	1.00	1.00	1681
Host injection	0.53	0.98	0.69	2545
Link fabrication	0.95	0.99	0.97	1727
Normal	1.00	0.99	0.99	216,746
Port scan	1.00	1.00	1.00	947
Accuracy			0.99	225,012
Macro Avg	0.91	0.99	0.94	225,012
Weighted Avg	0.99	0.99	0.99	225,012
Traffic-Only (29 features)				
DDoS	1.00	0.99	1.00	1366
Port hijack	1.00	1.00	1.00	1681
Host injection	0.37	0.99	0.54	2545
Link fabrication	0.11	0.99	0.19	1727
Normal	1.00	0.91	0.95	216,746
Port scan	0.99	1.00	1.00	947
Accuracy			0.92	225,012
Macro Avg	0.74	0.98	0.78	225,012
Weighted Avg	0.99	0.92	0.94	225,012
OpenFlow-Only (11 features)				
DDoS	0.84	0.99	0.91	1366
Port hijack	0.18	0.96	0.30	1681
Host injection	0.22	0.93	0.35	2545
Link fabrication	0.52	0.97	0.68	1727
Normal	1.00	0.91	0.95	216,746
Port scan	0.27	0.90	0.41	947
Accuracy			0.91	225,012
Macro Avg	0.50	0.94	0.60	225,012
Weighted Avg	0.98	0.91	0.94	225,012

3.5. Data Refinement and De-Identification

To ensure that the dataset is optimized for machine learning benchmarking, and to prevent classifier overfitting, a de-identification and cleaning process was applied to the raw flows. First, all network-layer identifiers—specifically, source and destination MAC and IP addresses—were removed. This prevents models from performing “shortcut learning” specific attacks. The structural integrity of the experiments is maintained through the topology and `last_layer` features, which provide the necessary architectural context without exposing specific host addresses.

Furthermore, the dataset was treated for protocol-specific null values to ensure a “plug-and-play” format for standard computational frameworks. Features that are logically

absent in certain protocols, such as `arp_opcode` for non-ARP traffic, were filled with a value of 0. Statistical features that could not be calculated for a single packet were also normalized to 0 to represent a lack of measurable variance or rate. This cleaning process ensures that researchers can utilize the full scale of 1,125,059 records without encountering execution errors in common machine learning libraries. Cleaning is applied at consolidation, identically to all files. A zero denotes “not applicable” exactly when `arp_opcode` accompanies an `eth_type` other than 0x0806, or when `packets` equals 1.

In addition to de-identification, GSO/TSO coalescing on the OVS virtual interfaces can merge segments into frames exceeding the 1500 B Ethernet MTU, producing `packet_size_max` values above the physical limit. We quantified the extent of this effect. A total of 5038 flows (0.45% of the dataset) carry `packet_size_max` above the 1500 B MTU. All are TCP flows, and all belong to the normal class; no attack-class record is affected. The affected flows appear across all five topologies (linear: 38, mesh: 1309, star: 1851, subnets: 246, tree: 1594). Because the artifacts are confined to a fraction of normal flows and absent from every attack class, they cannot act as a shortcut toward any attack class.

3.6. Reproducibility

All scripts used for testbed orchestration, traffic generation, and post-processing are publicly available in the same repository as the dataset. This includes the orchestrator script, the topology configuration files, traffic generators (both normal and attacks), and the flow-building and OpenFlow correlation pipeline. Researchers can use these scripts to replicate the experiments or generate additional data under different topologies or attack configurations.

4. Technical Validation

Normal traffic constitutes 96.3% of records. The imbalance was designed to provide a more demanding benchmark than a balanced dataset; it is not presented as reproducing attack prevalence in any specific deployment. Representation is comparable across topologies for all classes except link fabrication in the tree topology, which contains 196 records against a mean of approximately 2110 in the remaining four, as presented in Table 2. This deviation is discussed in Section 4. This section presents UMAP-based class separability analysis and an ablation study, following the precedent of [19] in evaluating the contribution of OpenFlow features to multi-class detection.

Firstly, Figure 8 presents a UMAP embedding computed on up to 5000 flows per class, capped at class size (port scan: 4737), RobustScaler-normalized with packet sizes clipped to 1500 B; `n_neighbors` = 30, `min_dist` = 0.1, Euclidean, seed 42”. Six distinct regional groupings are visible, consistent with the 40-feature representation providing a class-discriminative structure despite the nonlinear nature of the separation. DDoS, hijack, and port scan form compact, well-separated clusters, consistent with their strong classifier performance ($F1 = 1.00$, 1.00 , and 1.00 respectively). In contrast, link fabrication flows arrange as an elongated stripe structure rather than a compact cluster, reflecting continuous variation in forged LLDP injection rate across sessions. On the other hand, normal flows appear predominantly in a distinct region with scattered points near attack boundaries, attributable to the protocol overlap between legitimate ARP traffic and Injection flows.

Host injection is the only class exhibiting marked fragmentation into multiple sub-clusters. This fragmentation is not topology-driven, since all five topologies are uniformly distributed across every subcluster. It is also not attributable to a single feature, suggesting combinatorial behavioral variation across attack sessions. This finding is consistent with the lowest per-class $F1$ score observed during classifier validation ($F1 = 0.69$) and motivates future work on injection subtype characterization.

Secondly, the dataset was used to experiment with an XGBoost model as part of the validation. The hyperparameters used were 500 estimators, maximum depth of 8, and learning rate of 0.05, with subsampling ratios of 0.8 for both instances and features per tree. Categorical features were ordinally encoded and rate-based features scaled with RobustScaler prior to training. The dataset was partitioned into 80% training and 20% test sets using stratified splitting to preserve class proportions. This classifier serves exclusively as a validation instrument to quantify the discriminative utility of the feature set; it does not constitute a proposed detection method. Hyperparameters were fixed a priori and not tuned against test performance; no class weighting was applied. Library versions and the random seed are given in `environment.md`.

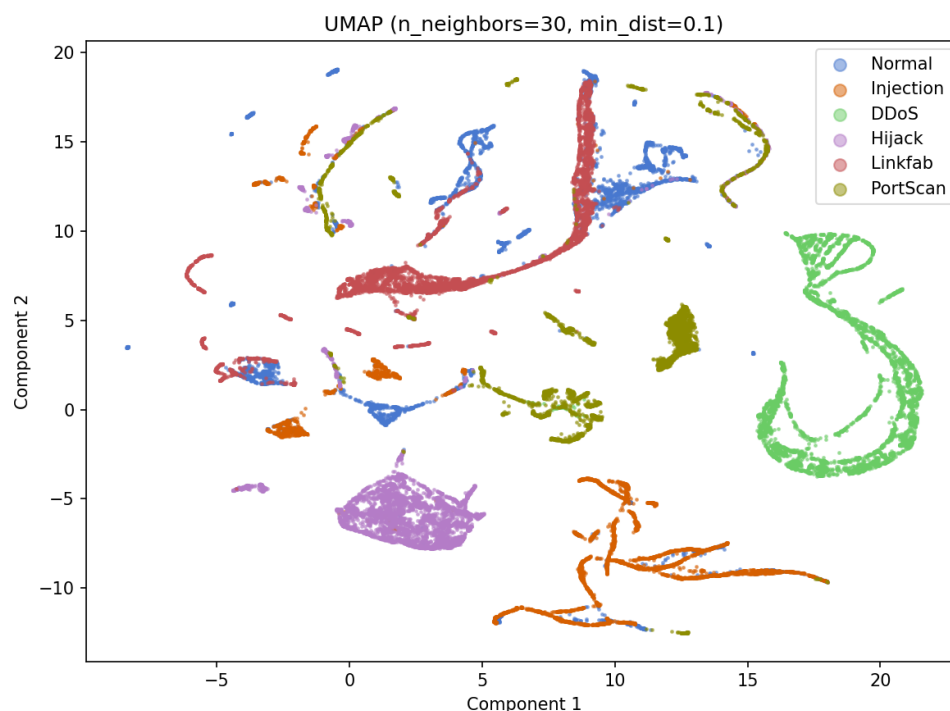


Figure 8. UMAP (Uniform Manifold Approximation and Projection) visualization of the dataset using a stratified sample of 5000 records per class.

Table 8 reports the classifier performance under three feature conditions: full feature set (40 features), traffic-only (29 features, OpenFlow metrics excluded), and OpenFlow-only (11 features). The full feature set achieved a macro F1 of 0.94. The ablation results indicate that the eleven OpenFlow control-plane features are decisive under the conditions evaluated. However, when OpenFlow features were removed, the macro F1 reduced to 0.78, with the most pronounced degradation observed in link fabrication detection, where F1 collapsed from 0.97 to 0.19, as presented in Figure 9. This collapse occurred because link fabrication attacks produce minimal statistical deviation in traffic-level features but generate anomalous OpenFlow messaging patterns that are only captured by control-plane metrics. The OpenFlow-only condition achieved macro F1 of 0.60, confirming that control-plane features alone are insufficient but provide complementary discriminative signal.

These results demonstrate that the OpenFlow control-plane features are decisive under the models and conditions evaluated here, particularly for detecting attacks that exploit SDN architectural mechanisms without generating anomalous traffic-level signatures.

The injection class exhibits the lowest per-class precision (0.53), driven primarily by 2243 normal flows misclassified as injection (1.0% of normal test samples), rather than by missed injection detections (recall = 0.98). Both classes share ARP as the exclusive protocol

(eth_type = 0x0806), making traffic-level feature separation structurally limited. This overlap is a dataset characteristic rather than a modeling artifact, and it motivates future work on finer-grained ARP behavioral features for injection detection.

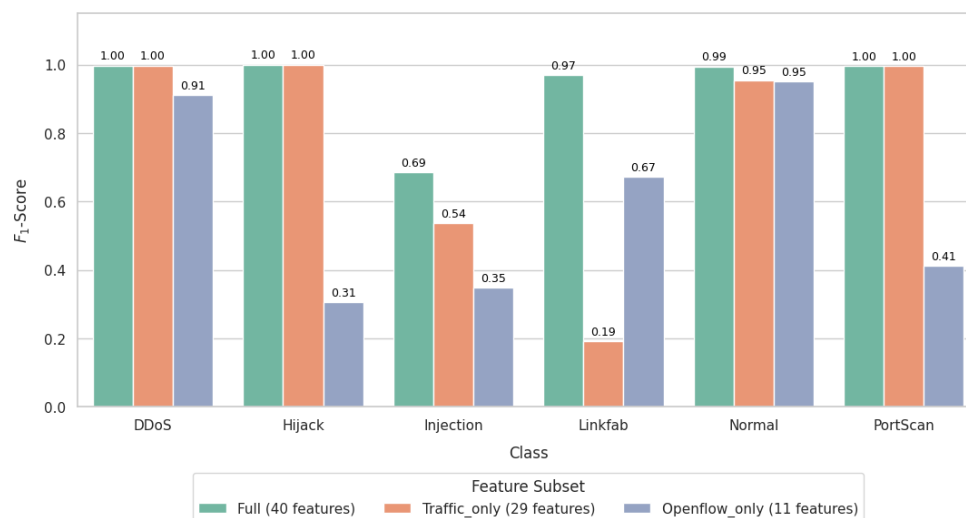


Figure 9. Per-class F1-score comparison across feature subsets.

Limitations and Scope

The dataset prioritizes reproducibility, which motivated a fully containerized, single-machine testbed with traffic simulators rather than real applications. Consequently, several constraints define its scope. All experiments ran under a single controller: OpenDaylight was selected for its OpenFlow 1.3 support and mature southbound telemetry interface; nevertheless, behavior under alternatives such as ONOS or Ryu remains untested. Moreover, traffic was synthetically generated, so it reproduces protocol structure but not the full statistical richness of real user behavior. Additionally, the environment was executed on one host machine, and switching relied on Open vSwitch virtual interfaces rather than physical hardware; as a result, packet-size features are subject to GSO/TSO coalescing artifacts and timing reflects virtualized dynamics.

Further constraints arise from the experiment design. Because some attacks degrade controller performance, they were executed in a fixed sequence with fixed cooldown gaps, scheduling the most disruptive ones last to avoid contaminating subsequent captures. Consequently, execution-stage effects cannot be separated from attack class within a single run; labels, however, derive from the reserved-MAC oracle rather than from timing. Furthermore, each attack class relies on a single implementation mechanism, which limits intra-class variant diversity.

Constraints also apply to the evaluation. Each topology was generated from a single run; therefore, per-topology variance cannot be estimated, and reported metrics should be read as point estimates. Moreover, cross-topology leave-one-out validation was not performed, so the reported split measures feature discriminability within the released data rather than generalization to unseen topologies; the per-topology files are released to enable it. One anomaly is documented: the tree topology yielded 196 link fabrication flows against a mean of approximately 2110 in the remaining four (Table 2). The cause has not been isolated; per-topology link fabrication analyses on that topology are therefore not supported by adequate sample size. Similarly, a per-feature ablation within the control-plane block was not performed; the released training script permits it.

In addition, feature values depend on the flow-building timeouts and on the 10 ms pad applied to sub-millisecond flows (Section 3.3), and would therefore shift if these were changed.

Finally, the traffic scope is bounded to unencrypted LAN conditions, without encrypted payloads, WAN effects, or cross-site topologies. However, the complete generation pipeline is released, so the experiments can be repeated; an independent reproduction was not performed. Researchers extending the dataset beyond these conditions should validate model generalization accordingly.

5. User Notes

The dataset is publicly available at e-cienciaDatos under DOI 10.21950/QMAXKP and is distributed under a Creative Commons Attribution 4.0 International (CC BY 4.0) license, permitting unrestricted reuse with attribution.

For machine learning applications, `FULL_SDN_NIDS.csv` and `FULL_SDN_NIDS.parquet` are the recommended files, as they consolidate five topologies (1,125,059 records). The five individual topology files are provided for topology-specific studies or cross-topology generalization experiments. The `label` column stores abbreviated class strings (Normal, Injection, Linkfab, Hijack, DDoS, PortScan); the mapping to the full class names used throughout this descriptor is given in Table 6.

Prior to model training, users should account for three practical considerations: First, the dataset is heavily class-imbalanced: normal traffic constitutes 96.3% of records; stratified train/test splitting and class-weighted loss functions are recommended. Second, categorical features (`last_layer`, `eth_type`, `Ofpredominant_type`, `topology`) require encoding before use with numerical classifiers. Third, packet-size features (`packet_size_max`, `packet_size_min`, `packet_size_range`) are subject to GSO/TSO coalescing artifacts on the OVS virtual interfaces, quantified in Section 3.5; values exceeding the 1500-byte Ethernet MTU should be clipped prior to analysis.

The dataset does not contain personally identifiable information. All traffic was synthetically generated in an emulated environment; no ethical restrictions apply to its use or redistribution.

6. Conclusions

This dataset provides a reproducible, multi-class benchmark in which SDN-architectural attacks—host injection (ARP), link fabrication (LLDP), and port hijack—are captured alongside the conventional DDoS and port scan classes. The ablation results confirm that the eleven OpenFlow control-plane features are decisive for the most SDN-specific classes: without them, link fabrication detection collapses ($F1\ 0.97 \rightarrow 0.19$) and host-injection detection degrades markedly. The UMAP supports class separability for all classes except host injection, which overlaps with normal traffic because injected and legitimate ARP exchanges share the same protocol signature (`eth_type 0x0806`); sharpening this boundary remains open work.

Several limitations bound these results and should guide reuse. The testbed uses a single controller (OpenDaylight), so behavior under ONOS or Ryu is untested. Traffic is synthetic and runs on a single virtualized host: packet-size features carry GSO/TSO coalescing artifacts, and timing reflects virtualized rather than physical dynamics. Each attack class relies on one implementation mechanism, limiting intra-class variant diversity, and each topology derives from a single run, so reported metrics are point estimates without per-topology variance. Feature values also depend on the flow-building timeouts and the sub-millisecond pad. Finally, the scope is restricted to unencrypted LAN conditions, excluding encrypted payloads, WAN effects, and cross-site topologies. All experiments

remain reproducible from the provided scripts, and researchers extending the dataset should re-validate model generalization accordingly.

Author Contributions: Conceptualization, T.Z.-O.; methodology, J.A.-H.; software, J.A.-H.; validation, J.M.R.; formal analysis, J.A.-H.; investigation, J.A.-H.; writing—original draft preparation, J.A.-H.; writing—review and editing, J.I.M.T.; supervision, J.E.C. and J.I.M.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The dataset and all generation, labeling, and analysis code are publicly available at <https://doi.org/10.21950/QMAXKP>. The record includes the orchestrator, topology managers, traffic and attack generators, the flow-building and OpenFlow correlation pipeline, environment.md, and data_dictionary.csv.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SDN	Software-Defined Networking
NIDS	Network Intrusion Detection System
OF	OpenFlow
DDoS	Distributed Denial of Service
ARP	Address Resolution Protocol
LLDP	Link Layer Discovery Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
ICMP	Internet Control Message Protocol
MAC	Media Access Control
IP	Internet Protocol
ML	Machine Learning
UMAP	Uniform Manifold Approximation and Projection
F1	F1-Score (harmonic mean of precision and recall)
CSV	Comma-Separated Values
OSI	Open Systems Interconnection
MTU	Maximum Transmission Unit
GSO	Generic Segmentation Offload
TSO	TCP Segmentation Offload
OVS	Open vSwitch
CC BY	Creative Commons Attribution License
DOI	Digital Object Identifier

References

- McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 69–74. [[CrossRef](#)]
- Ahuja, N.; Singal, G.; Mukhopadhyay, D.; Kumar, N. Automated DDOS attack detection in software defined networking. *J. Netw. Comput. Appl.* **2021**, *187*, 103108. [[CrossRef](#)]
- Wang, Y.; Wang, X.; Ariffin, M.M.; Abolfathi, M.; Alqhatani, A.; Almutairi, L. Attack detection analysis in software-defined networks using various machine learning method. *Comput. Electr. Eng.* **2023**, *108*, 108655. [[CrossRef](#)]
- Bahashwan, A.A.; Anbar, M.; Manickam, S.; Issa, G.; Aladaileh, M.A.; Alabsi, B.A.; Rihan, S.D.A. HLD-DDoSDN: High and low-rates dataset-based DDoS attacks against SDN. *PLoS ONE* **2024**, *19*, e0297548. [[CrossRef](#)] [[PubMed](#)]
- Karmous, N.; Jlassi, W.; Aoueilayine, M.O.E.; Filali, I.; Bouallegue, R. A New Dataset for Network Flooding Attacks in SDN-Based IoT Environments. *Comput. Model. Eng. Sci.* **2025**, *145*, 4363. [[CrossRef](#)]

6. Alzahrani, A.O.; Alenazi, M.J. Designing a network intrusion detection system based on machine learning for software defined networks. *Future Internet* **2021**, *13*, 111. [\[CrossRef\]](#)
7. He, X.; Sun, S. A two-phase detection method against APT attack on flow table management in SDN: X. *J. Supercomput.* **2023**, *79*, 15415–15434. [\[CrossRef\]](#)
8. Elsayed, M.S.; Le-Khac, N.A.; Jurcut, A.D. InSDN: A novel SDN intrusion dataset. *IEEE Access* **2020**, *8*, 165263–165284. [\[CrossRef\]](#)
9. Yasarathna, T.L.; Le-Khac, N.A. ASEADOS-SDN-IoT: A novel SDN-IoT network intrusion detection dataset and framework. *Internet Things* **2026**, *36*, 101891. [\[CrossRef\]](#)
10. Ahuja, N.; Singal, G.; Mukhopadhyay, D.; Nehra, A. Ascertain the efficient machine learning approach to detect different ARP attacks. *Comput. Electr. Eng.* **2022**, *99*, 107757. [\[CrossRef\]](#)
11. Wang, S.; Balarezo, J.F.; Chavez, K.G.; Al-Hourani, A.; Kandeepan, S.; Asghar, M.R.; Russello, G. Detecting flooding DDoS attacks in software defined networks using supervised learning techniques. *Eng. Sci. Technol. Int. J.* **2022**, *35*, 101176. [\[CrossRef\]](#)
12. Tavallaee, M.; Bagheri, E.; Lu, W.; Ghorbani, A.A. A Detailed Analysis of the KDD CUP 99 Data Set. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*; IEEE: New York, NY, USA, 2009; pp. 1–6. [\[CrossRef\]](#)
13. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A.A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* **2018**, *1*, 108–116. [\[CrossRef\]](#)
14. Buzzio-García, J.; Vergara, J.; Ríos-Guiral, S.; Garzón, C.; Gutiérrez, S.; Botero, J.F.; Quiroz-Arroyo, J.L.; Pérez-Díaz, J.A. Exploring Traffic Patterns Through Network Programmability: Introducing SDNFlow, a Comprehensive OpenFlow-Based Statistics Dataset for Attack Detection. *IEEE Access* **2024**, *12*, 42163–42180. [\[CrossRef\]](#)
15. Pal, G.; Dey, R.; Shankar, A.; Punugoti, C.S.; A, H.I.; Singh, M.P.; Lal, S. *CSDN: A Comprehensive Dataset for Attack Detection and Classification in SDN*; IEEE Dataport: New York, NY, USA, 2026. [\[CrossRef\]](#)
16. Peuster, M.; Karl, H.; Van Rossem, S. MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments. In *Proceedings of the 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*; IEEE: New York, NY, USA, 2016; pp. 148–153.
17. Li, J.; Qin, S.; Tu, T.; Zhang, H.; Li, Y. Packet Injection Exploiting Attack and Mitigation in Software-Defined Networks. *Appl. Sci.* **2022**, *12*, 1103. [\[CrossRef\]](#)
18. Soltani, S.; Shojafar, M.; Mostafaei, H.; Pooranian, Z.; Tafazolli, R. Link Latency Attack in Software-Defined Networks. In *Proceedings of the 2021 17th International Conference on Network and Service Management: Smart Management for Future Networks and Services, CNSM 2021*; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2021; pp. 187–193. [\[CrossRef\]](#)
19. Pansari, N.; Srivastava, S.; Agarwal, M. Attack classification using machine learning on UNSW-NB 15 dataset using XGBoost feature selection & ablation analysis. In *Proceedings of the 2024 IEEE 9th International Conference for Convergence in Technology (I2CT)*; IEEE: New York, NY, USA, 2024; pp. 1–9.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.