



This is a peer-reviewed, final published version of the following in press document and is licensed under Creative Commons: Attribution 4.0 license:

**H. Ahmed, Azza, Rukh, Mah ORCID logoORCID:
<https://orcid.org/0000-0001-7660-1150> and Arsalaan, Ameer
Shakayb (2026) In-Network Security Framework for Detecting
and Classifying DDoS Attacks. IEEE Access. p. 1.
[doi:10.1109/ACCESS.2026.3745922](https://doi.org/10.1109/ACCESS.2026.3745922) (In Press)**

Official URL: <https://doi.org/10.1109/ACCESS.2026.3745922>

DOI: <http://dx.doi.org/10.1109/ACCESS.2026.3745922>

EPrint URI: <https://eprints.glos.ac.uk/id/eprint/16601>

Disclaimer

The University of Gloucestershire has obtained warranties from all depositors as to their title in the material deposited and as to their right to deposit such material.

The University of Gloucestershire makes no representation or warranties of commercial utility, title, or fitness for a particular purpose or any other warranty, express or implied in respect of any material deposited.

The University of Gloucestershire makes no representation that the use of the materials will not infringe any patent, copyright, trademark or other property or proprietary rights.

The University of Gloucestershire accepts no liability for any infringement of intellectual property rights in any material deposited but will remove such material from public view pending investigation in the event of an allegation of any such infringement.

PLEASE SCROLL DOWN FOR TEXT.

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

In-Network Security Framework for Detecting and Classifying DDoS Attacks

AZZA H. AHMED^{1 2}, MAH-RUKH FIDA³, AND AMEER SHAKAYB ARSALAAN.⁴

¹Department of Computer Science, Oslo Metropolitan University, Norway

²Simula Metropolitan Center for Digital Engineering, Norway

³School of Business, Computing and Social Sciences, University of Gloucestershire, UK

⁴Department of Computer Science, University of Swabi, Pakistan

Corresponding author: Azza H. Ahmed (e-mail: azzahass@oslomet.no).

ABSTRACT The rapid proliferation of resource-constrained Internet of Things (IoT) devices has significantly expanded the network attack surface, increasing vulnerability to Distributed Denial of Service (DDoS) attacks. Existing programmable data-plane defenses either detect malicious flows without attributing the specific attack type, or achieve fine-grained attribution only through control-plane processing that reintroduces the latency in-network defenses aim to avoid. This paper introduces *SecFlow*, a dual-stage defense framework that performs real-time DDoS detection directly in the P4 data plane and attributes the specific attack type at a backend server, covering nine subtypes spanning volumetric, protocol-based, and application-layer attacks, enabling rapid, precise updates to switch-level mitigation rules. Evaluated on the CICIoT2023 dataset, the recommended TCP- and UDP-specific decision-tree detectors achieve flow-level accuracy of up to 99% for TCP and a perfect flow-level true-negative rate with 86% true-positive rate for UDP. A ResNet-based backend classifier attributes the specific attack type with 95% weighted accuracy (macro-F1 0.88); per-class F1-scores reach 0.95–1.00 for volumetric attacks and 0.82–0.99 for protocol-based attacks, but fall to 0.43 (HTTP Flood) and 0.86 (Slowloris) for application-layer attacks, reflecting feature overlap among low-rate attack types. In a separate, harmonized comparison against four representative in-network and hybrid DDoS detectors on a common Docker-based P4 testbed – using a coarser traffic grouping needed for fair cross-system evaluation – *SecFlow* is the only system that sustains strong flow accuracy (≥ 0.86) on both benign and malicious traffic across every evaluated category, while operating at the second-lowest memory footprint and among the lowest CPU usage of the five systems tested; every comparative system exhibits a severe accuracy collapse on at least one category, including benign traffic in one case. Implemented on Mininet with software P4 switches, *SecFlow* avoids the 3.72 ms median round-trip delay a controller-based baseline requires for every packet, reducing detection latency by one to two orders of magnitude, while cutting controller-bound link load by up to 67% relative to that baseline at a modest resource cost on the classifying switch itself. Together, these results show that fine-grained, per-attack-type DDoS defense is achievable at the data plane without sacrificing the responsiveness that motivates in-network detection in the first place.

INDEX TERMS Cyber-attack, detection, P4, switch, machine learning, network, programmable data-plane, intrusion detection, entropy

I. INTRODUCTION

DDoS attacks remain one of the most persistent and rapidly evolving threats to Internet security. Recent industry reports show that both the frequency and the scale of these attacks continue to grow sharply: Cloudflare reported mitigating 47.1 million DDoS attacks in 2025, including a record 31.4 Tbps volumetric attack and HTTP DDoS campaigns exceeding 200 million requests per second, with network-layer attack

volume more than tripling year-on-year (1). Nokia further observed that 2025 marked a shift towards more automated and precisely orchestrated DDoS campaigns, including attacks coordinated across tens of thousands of compromised IoT devices (2). Taken together, these trends indicate that modern DDoS attacks are no longer limited to simple volumetric flooding, but increasingly combine network-layer, transport-layer, and application-layer techniques at growing scale and

automation – making early, line-rate, resource-aware detection more critical than ever.

Traditional defenses, including static IP blacklists/whitelists (3; 4) and statistical methods such as entropy-based analysis (5; 6) or sketch/threshold-based heavy-flow tracking (7; 8; 9; 10) – struggle to keep pace with these evolving attack strategies. Machine Learning (ML)- and Artificial Intelligence (AI)-based Intrusion Detection Systems (IDS) (11) improve detection accuracy but face scalability and resource-demand challenges at large-network scale, and centralized IDS deployments compound these costs with added latency and network overhead, slowing detection and response (12).

Software Defined Network (SDN) (13) offers a promising defense architecture: by separating the control and data planes (14), it grants operators a global, programmable view of the network, empowering researchers to deploy lightweight, line-rate defense logic directly on switches. DDoS attacks broadly fall into three categories (15): volumetric attacks, which overwhelm bandwidth with high-rate traffic; protocol-based attacks, which exploit Layer 3/4 vulnerabilities to exhaust target resources; and application-layer attacks, which exploit Layer 7 vulnerabilities to mimic legitimate requests. The boundary between these categories is not strict, as an attack can exhibit characteristics of more than one depending on the network layer it targets.

Existing in-network and SDN-based DDoS defenses address only part of this problem. Sketch- and threshold-based data-plane detectors (7)– (10) identify heavy flows at line rate but cannot attribute attack type, forcing generic, uniform mitigation. Control-plane and hybrid ML-based classifiers (16; 17; 18) attribute attack type more richly but incur a per-packet round-trip delay to the controller – 3.72 ms in median in our own testbed measurements (Section V) – eroding the latency advantage that motivates in-network defense in the first place. Other data-plane ML approaches either detect a narrow subset of attacks (e.g., only SYN or only low-and-slow floods) (19; 20), or collapse detection to a binary attack/benign decision without distinguishing DDoS subtypes (21; 22). None combines (i) sub-millisecond, TCP- and UDP-specific binary detection in the data plane – one to two orders of magnitude faster than the controller round-trip delay above – (ii) fine-grained attack-type attribution that generalizes across the full spectrum of volumetric, protocol-based, and application-layer DDoS categories – spanning nine distinct attack types – at a backend server driven by selectively mirrored packets rather than full-flow mirroring, and (iii) a feedback path that makes this attribution available to the controller for per-type mitigation, rather than only a generic attack/benign signal. This paper introduces *SecFlow*, a dual-stage in-network defense framework that closes this gap by combining line-rate, TCP- and UDP-specific attack detection in the P4 data plane – using a tree-based machine learning model expressed in the P4 language (23) – with fine-grained attack-type attribution at a backend server reachable via the control plane. To the best of our knowledge, *SecFlow* is the first in-network DDoS defense to unify these three properties

within a single architecture, and its evaluation further provides – for the first time in this line of work – a quantitative, same-testbed comparison against four representative prior systems, rather than the qualitative feature-scope comparisons typical of prior surveys.

The key contributions of this study are as follows:

- **Dual-stage in-network DDoS defense:** Unlike prior systems that force a choice between fast, coarse in-switch detection and accurate but slow control-plane attribution, we design *SecFlow*, a two-stage framework that performs binary malicious-flow detection directly in the P4 data plane and defers fine-grained attack-type attribution to a backend ResNet classifier – achieving both line-rate detection and rich attack-type attribution without the controller round-trip latency of prior ML-based IDS.
- **TCP- and UDP-specific, lightweight data-plane detectors:** We design and implement separate decision-tree detectors for TCP and UDP traffic as native P4 match-action pipelines, recovering detection accuracy that a single combined detector loses under live P4 timing conditions. We further provide, to our knowledge for the first time, a quantitative characterization of the resulting trade-off between per-protocol feature economy and the switch's aggregate match-action rule footprint, offering concrete guidance for future lightweight in-network detector design.
- **Selective, bandwidth-aware mirroring for attack attribution:** Rather than mirroring entire malicious flows to the controller, *SecFlow* forwards only a sampled subset of packets with truncated payloads, cutting controller-bound link load by up to 67% while preserving classification accuracy.
- **Comprehensive, same-testbed empirical evaluation:** We evaluate *SecFlow* on the CICIOT2023 dataset across nine DDoS subtypes spanning volumetric, protocol-based, and application-layer attacks on a Mininet/BMv2 P4 testbed, and – addressing a gap left by the largely qualitative comparisons in prior work – benchmark *SecFlow* against four representative in-network and hybrid DDoS detection systems under identical conditions to quantify comparative accuracy and resource efficiency.

These contributions are validated experimentally throughout the paper: the TCP- and UDP-specific detectors achieve up to 99% flow-level accuracy for TCP (D_{wo}^T) and a perfect flow-level TNR with 86% TPR for UDP (D_{wo}^U), avoiding the collapse to as low as 0.55 flow-level F1 on UDP traffic exhibited by a single, combined, time-inclusive detector (D_w); the backend classifier reaches 95% weighted accuracy across nine attack types; the combined pipeline limits overhead to $\leq 11\%$ CPU and under 230 MB RSS on the software switch while reducing controller-bound link load by up to 67% relative to a standard IDS; and, benchmarked against four representative baseline systems on a common testbed, *SecFlow* is the only one to sustain flow accuracy ≥ 0.86 on both benign and malicious traffic across every evaluated

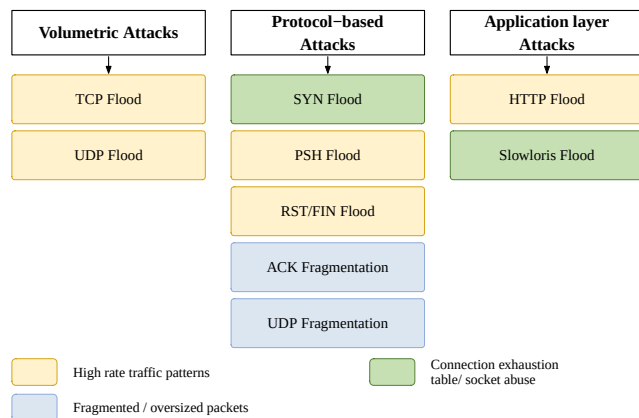


FIGURE 1. Hierarchical classification of DDoS attacks investigated in this study.

category.

The paper is structured into seven additional sections. After discussing the study's background in Section II, Section III introduces the *SecFlow* framework for detecting and classifying DDoS attacks. Section IV outlines the experimental setup, dataset used, and features selection process. In Section V, we provide a detailed results analysis of the attack detection and classification process within the dual-stage framework, followed by a discussion of *SecFlow* performance improvements and limitations in Section VI. A review of related literature is presented in Section VII, and the paper concludes with insights into future research directions in Section VIII.

II. BACKGROUND

In this section, we present an overview of the three categories of DDoS attacks targeted for attribution during traffic flow analysis. We also introduce a distributed SDN network, which we assume to serve as the foundation for scalability of the defense systems like *SecFlow*. Additionally, we outline the operational pipeline of P4, a programmable data plane language.

A. TAXONOMY OF DDOS ATTACKS

DDoS attacks aim to disrupt the normal traffic of a targeted server, service, or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic. Although a single type of DDoS attack may exhibit characteristics of multiple categories, we classify each attack according to the most appropriate category defined by (24). They broadly categorize DDoS attacks into volumetric, protocol, and application-layer types. Figure 1 illustrates how the attacks considered in this study are grouped within these three categories and also highlights cross-category attacks that share similar characteristics.

- 1) **Volumetric Attacks:** These attacks generate an immense volume of traffic, overwhelming the target's bandwidth. The vast quantity of traffic can impede

access to resources such as websites or services. Such attacks are often orchestrated using botnets composed of compromised devices. The scale of these attacks is typically measured in bits or packets per second, with common types including UDP Flood and TCP Flood.

- 2) **Protocol-based Attacks:** These attacks target vulnerabilities within the layer 3 or layer 4 protocol stack, specifically focusing on exhausting the processing capabilities of the target system or critical network infrastructure such as firewalls, resulting in significant service disruption. The intensity of such attacks is quantified in packets per second (pps). Common layer 4 attacks exploit weaknesses in the TCP/UDP protocol, including:

- a) *SYN Flood:* Attackers flood a server with forged SYN requests, exploiting TCP's three-way handshake by never sending the final ACK. This creates many half-open sessions, causing connection exhaustion and socket abuse of SYN flood as the server holds resources (sockets, memory, bandwidth) waiting for replies that never come. The result is degraded performance or denial of service.
- b) *ACK Flood:* This attack overwhelms a server by bombarding it with a large volume of spoofed ACK packets that do not correspond to any legitimate TCP sessions. As the server attempts to match these bogus ACKs with non-existent SYN or SYN-ACK packets, its CPU, memory, and connection tables become overloaded. This resource exhaustion leads to system slowdowns and degraded performance.
- c) *ACK Fragmentation Flood:* A variation of the ACK Flood, involves sending fragmented packets up to the maximum segment size of 1,500 bytes to exploit the maximum IP packet size of 65,535 bytes (including headers). The excessive reassembly demands can lead to memory overflow errors or crashes in servers and network equipment.
- d) *PSH Flood:* In this attack, once a connection is established, attackers send continuous streams of TCP segments marked with the PUSH flag, compelling the server to process each segment immediately. This can severely slow down or crash the server, disrupting service to legitimate users.
- e) *RST/FIN Flood:* Attackers send spoofed RST or FIN packets in large numbers, causing the server to expend resources attempting to associate these packets with non-existent TCP sessions.
- f) *UDP Fragmentation:* When an IP datagram is larger than the Maximum Transfer Unit (MTU) of the route, it is divided into fragments. In UDP fragmentation attack, the attacker therefore uses large UDP packets which forces its fragmentation in the network, leading to consumption of more

network bandwidth with fewer packets. Additionally, it has the potential to consume server CPU resources and fill memory buffers associated with the processing and reassembling of fragmented packets.

These attacks cleverly abuse protocol behaviors to deplete server resources, often leading to significant downtime and disruption.

- 3) **Application Layer Attacks:** Exploiting weaknesses in the Layer 7 protocol stack, certain attacks pose substantial detection challenges. These attacks include tactics like low-and-slow approaches, GET/POST floods, and specific exploits targeting systems such as Apache, Windows, or OpenBSD. These operations use requests that seem legitimate and harmless to crash web servers, with their intensity quantified by Requests per Second (rps) (25).

Identifying these threats can be particularly challenging as the attacks mimic legitimate user behaviors. Thus, they often result in only slight increases in traffic, without requiring the deployment of a botnet. The prevalent attacks include:

- a) *HTTP Flood*: This attack involves overwhelming websites and servers by misusing HTTP commands. Variants of HTTP Flood include GET Attacks, POST Attacks, Low-and-Slow POST Attacks and Fragmented HTTP Flood.
- b) *Slowloris*: It targets web servers by sending incomplete HTTP requests and keeping numerous connections open for as long as possible using minimal bandwidth. By continuously maintaining these half-open connections, the attacker prevents the server from closing them, leading to connection exhaustion and socket abuse.

While volumetric DDoS defenses typically rely on network-level techniques such as rate limiting and packet inspection to filter malicious traffic, application-level solutions analyze the content and behavior of requests to detect malicious intent beyond basic IP or header attributes. In this study, we explore whether a P4 switch can serve as a comprehensive, line-rate DDoS detector, capable of identifying various types of DDoS attacks across both network and application layers.

B. SEPARATION OF CONTROL AND DATA PLANE

Traditional network devices, such as routers and switches, integrate two key functional planes: the *control plane*, which makes decisions about packet handling and routing, and the *data plane*, which executes these decisions by forwarding packets at high speed.

The emergence of the Software-Defined Networking (SDN) paradigm (26) introduced a major architectural shift by decoupling the control plane from individual devices and implementing it as a logically centralized software controller. Hosted on a backend server, this controller manages network behavior programmatically, while data plane devices are

simplified into efficient packet-forwarding elements. This design grants a global network view, enabling flexible and dynamic management of flows and policies.

Such separation has proven highly effective in domains like cloud data centers and IoT networks, where centralized control facilitates rapid reconfiguration, scalability, and uniform policy enforcement. Furthermore, the SDN control plane supports advanced operations such as traffic engineering, load balancing, and Quality of Service (QoS) management (27). It also underpins security applications (28), including dynamic threat detection, malicious traffic attribution, and adaptive rule updates on data plane devices—enhancing responsiveness and resource optimization across the entire network.

Collectively, these components establish a unified and flexible framework that abstracts hardware complexity, enabling portable, high-level packet processing across diverse switching platforms. This programmability has become a cornerstone of SDN evolution, combining high performance with adaptability and intelligent network control.

C. PROGRAMMABLE DATA PLANE AND P4 LANGUAGE

Within the SDN paradigm, the data plane is tasked with realizing the forwarding and processing logic determined by the control plane. It operates at high speed, executing functions such as header parsing, field matching, and packet forwarding. The emergence of programmable data planes has significantly advanced this capability, allowing operators to redefine packet-processing behavior dynamically. This flexibility fosters rapid innovation by enabling the in-network deployment of customized mechanisms for security enforcement, traffic engineering, load balancing, and performance monitoring.

The *Programming Protocol-Independent Packet Processors* (P4) language (29) was introduced to enable protocol-independent and target-agnostic control over packet processing within the data plane. P4 allows developers to specify how packets are parsed, processed, and forwarded, including the definition of custom headers, parsing logic, and match-action behaviors. It has emerged as the de facto standard for programming Reconfigurable Match-Action Table (RMT) architecture-based switches. The P4 compiler, known as *P4C*, supports multiple target architectures, with the widely used *v1Model* (30) being one of the most common. The *v1Model* defines a logical pipeline composed of four primary components (see Figure 2):

- **Programmable Parser:** Extracts relevant header fields from incoming packets based on programmable parsing logic defined by the operator, supporting diverse and custom protocol structures.
- **Ingress Pipeline:** Constitutes the core processing unit, organized into sequential match-action stages. Each stage performs lookups in memory tables and executes corresponding actions—such as forwarding, modification, or dropping—using Arithmetic Logic Units (ALUs). The packet is then directed toward the appropriate egress port.

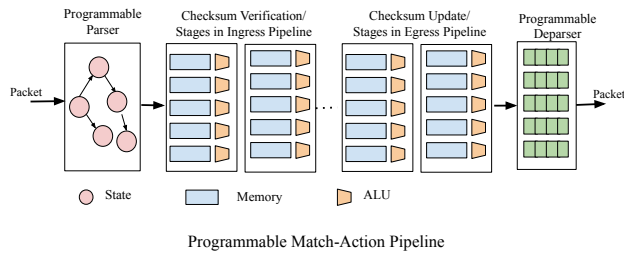


FIGURE 2. v1Model Switch Architecture.

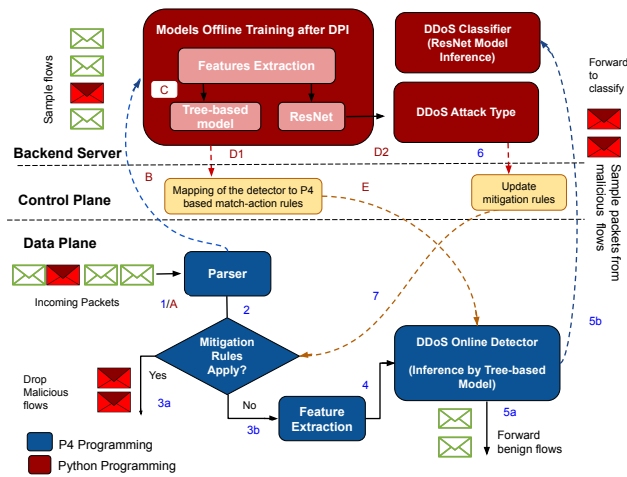


FIGURE 3. SecFlow architecture.

- **Egress Pipeline:** Processes packets retrieved from the egress port queue. It mirrors the functional design of the ingress pipeline, with compiler-level resource management ensuring that ingress and egress operations proceed without contention.
- **Programmable Deparser:** Reconstructs the processed packet by reassembling headers and payload into a valid frame suitable for transmission.

III. SYSTEM ARCHITECTURE

In this section, we introduce SecFlow, the proposed in-network defensive framework illustrated in Figure 3. Building upon the SDN principles of centralized control and programmable forwarding, SecFlow integrates lightweight detection at the data plane with high-fidelity classification at a backend security server. The controller and server form the system's logical control plane: the backend classifier's attack-type attribution provides the basis on which the controller can, in principle, coordinate network-wide decisions and update switch behavior in response to detected threats, though this study evaluates the detection and attribution stages but not the resulting switch-side enforcement.

A. ROLE AT DATA PLANE

The data plane in SecFlow is realized on a P4-programmable switch that performs real-time flow monitoring and preliminary DDoS detection (see steps 1-5 in Figure 3). Unlike prior works that rely solely on controller-based analysis (16; 19), our approach embeds an online inference module directly within the packet-processing pipeline. This design minimizes control-channel congestion and latency by executing first-stage classification at line rate.

Since P4 devices provide limited arithmetic and memory resources (31; 32), complex models such as deep neural networks cannot be efficiently deployed on-chip. Therefore, SecFlow implements a decision-tree-based model that can be expressed using simple conditional logic (*if-else*) and mapped onto the match-action stages of the ingress pipeline. Each stage represents a node in the decision tree, with packet features evaluated sequentially until a leaf node verdict (*benign* or *malicious*) is reached. For ensembles such as Random Forests, multiple parallel pipelines correspond to individual trees, and their outputs are aggregated via a voting mechanism.

The data plane also includes a lightweight feature extractor, which computes flow-level metrics (e.g., packet rate, byte count, and inter-arrival intervals) using stateful registers. After a flow is identified as suspicious, selective packet mirroring is performed toward the backend for detailed classification. To reduce processing and bandwidth overhead, the system mirrors only every n^{th} packet (in this implementation, the 3rd packet) from each flagged flow, as shown in Figure 4. This modular split ensures that resource-intensive analysis does not burden the switch while maintaining low-latency detection at the network edge.

B. ROLE AT BACKEND SERVER

The backend server, hosting the SDN controller and security engine, serves as the global coordinator and high-capacity analytics node. The interaction between the controller and network packets, along with its functional role within SecFlow, is illustrated through steps A-E in Figure 3. It receives mirrored traffic samples from edge switches and retrains or redeploys models as network conditions evolve. To refine the initial detections, SecFlow employs a Residual Neural Network (ResNet) (33) to classify individual packets into their specific attack class (e.g., SYN flood, UDP fragmentation, etc.). The ResNet architecture was selected over simpler alternatives (MLP, standard CNN) because residual connections stabilize gradient flow across stacked Conv1D layers, enabling hierarchical extraction of local feature correlations from the packet feature vector, while batch normalization and dropout provide regularization effective under the class imbalance present in the dataset. The architecture and input configuration of the ResNet classifier are described in detail in the following section.

Drawing on the adaptive sampling strategy proposed by (16), only a subset of packets (every third flagged packet) is forwarded for deep inspection. This balances detection

```
// The first detected packet of a flow is reported
if (meta.attack_packet == 1) {
    meta.forward_controller = 1;
    // Let n be 3
    reg_nth_attack_pkts.write(meta.register_index,
        3);
}

// nth packet: multiple of 3
reg_nth_attack_pkts.read(meta.nth_attack_packet,
    meta.register_index);
if (meta.attack_packet == meta.nth_attack_packet)
{
    meta.forward_controller = 1;
    meta.nth_attack_packet = meta.
        nth_attack_packet + 3;
    reg_nth_attack_pkts.write(meta.register_index,
        meta.nth_attack_packet);
}
```

FIGURE 4. P4 code snippet restricting the number of malicious packets per flow reported to the controller.

accuracy with communication overhead, enabling scalable operation across distributed domains. When the ResNet module confirms an attack, the controller can update the corresponding switch tables via P4Runtime to quarantine the malicious source and propagate mitigation policies across the SDN domain; this study evaluates the detection and attribution stages preceding this step (Section V), while the resulting mitigation actions themselves are not independently measured (Section VI).

Through this cooperative design, SecFlow bridges the performance gap between programmable hardware and server-side intelligence. The data plane provides immediate, resource-efficient detection, while the control plane ensures global visibility and dynamic response—achieving both responsiveness and scalability in DDoS defense.

IV. EXPERIMENTAL SETUP

This section describes the experimental dataset, the feature selection process across different detectors and classifiers, and the mapping of the detectors to P4.

A. DDOS ATTACKS DATASET

To develop and evaluate our system, we employ the CICIOT2023 dataset (34). This dataset provides PCAP files that can be utilized for both training and replaying network traffic within a controlled simulation testbed. The data is systematically organized into distinct folders according to the type and characteristics of each attack, while benign traffic is stored separately to minimize the risk of mislabeling or confusion during preprocessing. Furthermore, CICIOT2023 encompasses a comprehensive spectrum of DDoS attack categories and variants, offering a diverse representation of malicious behaviors. These properties make it particularly suitable for evaluating and benchmarking IDSs across heterogeneous environments and detection methodologies.

The dataset includes traffic from 105 IoT devices comprising smart home appliances, IP cameras, sensors, and

TABLE 1. Details of the dataset used in the study.

Type	Training CSV Records (Packets, Flows)	Test PCAP (Packets, Flows)	Tool
Volumetric Attacks			
TCP Flood	(65k, 65k)	(2000, 1499)	hping3 (35)
UDP Flood	(81k, 343)	(4000, 22)	udp-flood (36)
Transport Layer			
Ack-Fragmentation	(4k, 4.2k)	(3139, 3069)	hping3 (35)
RST/FIN Flood	(136k, 130k)	(2000, 2000)	hping3 (35)
SYN Flood	(48k, 40k)	(2000, 1995)	hping3 (35)
PSH Flood	(105k, 105k)	(2000, 2000)	hping3 (35)
UDP Fragmentation	(3.9k, 77)	(1050, 13)	udp-flood (36)
Application Layer			
HTTP Flood	(25k, 3.2k)	(2000, 645)	golang-httpflood (37)
Slowloris	(63k, 6k)	(2000, 284)	slowloris (38)
Malicious Total	(530k, 356k)	(20k, 11.5k)	—
Benign Total	(98k, 21k)	(3905, 327)	—

microcontrollers. It features 33 distinct attack types, with a predominant focus on DDoS activity. Since this work aims to detect volumetric, protocol-based, and application-layer DDoS attacks (see Figure 1) using a unified in-network module, we selected nine representative malicious flow types spanning these three categories.

We extracted at least 5MB of PCAP files for benign traffic as well as each malicious type listed in Table 1 as the training dataset, and at least 1MB of PCAP files for testing. The training and testing PCAP segments are strictly disjoint at the flow, time-window, and device level, eliminating the risk of train/test leakage from the CICIOT2023 traces. The training PCAPs are converted into CSV feature sets using a Python preprocessing script (PCAP2CSV.py), from which the decision-tree detectors described in Section III are trained. The dataset contains a total of 356k malicious and 21k benign flows. The distribution of records across the training and testing phases is shown in Table 1.

B. FEATURE SELECTION

We deploy a lightweight Decision Tree (DT) classifier for detection, adopted as a single-tree instance of a Random Forest (RF); the sensitivity of accuracy to tree count and depth is examined separately in Section V. Unlike (34), our design excludes features prone to model failure when unseen attack flows appear over time. For example, (34) employs flow identifiers comprising (source IP, destination IP, source port, destination port, transport protocol) and packet generation timestamps from PCAP files. Since attacks of similar nature can occur across arbitrary (source IP, destination IP) pairs at any time, we retain only the transport-layer protocol feature. We also exclude features currently unsupported in P4 (version 16), such as variance or standard deviation of per-flow values.

When training a malicious flow detector in Python, the inter-packet duration is derived from the timestamps (in

```
// The comparative threshold for the maximum inter
// -packet duration
// at a tree node is multiplied by 1000 for
// alignment
feature = meta.max_iat; // feature at a tree node
th = th * meta.UNITS; // comparative value at
// the same node
```

FIGURE 5. Approximate alignment of time-related features in P4 script.

milliseconds) of consecutive packets in a flow. In contrast, P4 (specifically v1Model) calculates this metric using the `ingress_global_timestamp` metadata, which records the time (in microseconds) at which a packet begins ingress processing. To align the units, we introduce a conversion factor `meta.UNITS = 1000`, as shown in Figure 5. Because this timestamp reflects a packet's actual arrival time at the switch rather than its original PCAP capture time, inter-packet gaps computed on the switch can diverge from the PCAP-derived gaps used during training whenever PCAP replay does not perfectly reproduce the original capture's timing. This discrepancy is compounded by software-switch scheduling jitter and virtualization overhead, and may affect both the stability and the generalization capability of time-dependent features. Consequently, we train and evaluate both time-inclusive and non-time-based variants of each detector, so that the practical impact of this discrepancy can be assessed empirically rather than assumed.

After converting training PCAP traces to CSV format¹, we apply standard feature selection techniques by examining pairwise correlations among predictors. Highly correlated features are pruned by retaining one representative variable (e.g., `flow_bytes` replaces `sbytes` and `dbytes`). As discussed earlier, only features computable within P4 are considered. Given this limited subset, we adopt the *Backward Elimination* method (39), which iteratively removes non-contributing features until optimal model performance is achieved.

Since DDoS attacks (see Figure 1) generally utilize TCP or UDP transport protocols, we include a categorical feature `type` to distinguish between them. For malicious packet and flow detection, we evaluate six variants of a detector model: D_w , D_w^T , D_w^U and D_{wo} , D_{wo}^T , D_{wo}^U . The first group (D_w , D_w^T , D_w^U) includes time-based features, whereas the latter (D_{wo} , D_{wo}^T , D_{wo}^U) omits them. Specifically, D_w and D_{wo} detect both TCP- and UDP-based DDoS attacks; D_w^T and D_{wo}^T are trained solely on TCP attacks; and D_w^U and D_{wo}^U focus on UDP-based flows. As summarized in Table 2, the time-based detectors (D_w , D_w^T , D_w^U) utilize seventeen, thirteen, and nine features, respectively, while their non-time-based counterparts employ fifteen, thirteen, and seven.

For classification of detected malicious flows, we implement ResNet-based models: C_w , which incorporates time-based features, and C_{wo} , which omits them. The categorical feature `type` is one-hot encoded, enabling the classifier to

effectively differentiate between TCP and UDP attacks. The complete set of features for each model, including these classifiers, is provided in Table 2.

Table 3 reports the compiled match-action footprint of each variant on the P4 switch: the number of features used, the number of sequential match-action stages a packet may traverse, and the total number of compiled rules across all stages. Splitting a detector by protocol increases the total number of installed rules relative to its same-family combined counterpart, since both protocol trees must remain resident on the switch to serve either protocol; this holds for both the non-time-based family ($D_{wo}^T + D_{wo}^U$: 576 rules, versus D_{wo} : 551) and the time-based family ($D_w^T + D_w^U$: 834 rules, versus D_w : 727). This increase in static footprint does not, however, translate into additional per-packet processing depth: because a packet is routed to exactly one tree according to its transport protocol, a TCP packet traverses at most eleven match-action stages and a UDP packet at most nine, in both cases no more than a single combined detector's eleven stages.

C. TESTBED INFRASTRUCTURE

The evaluation is performed inside a single Docker container built from the SecFlow Dockerfile, hosting Mininet and pre-compiled BMv2 (`simple_switch`) instances for Switch 1, Switch 2, and Switch 3, orchestrated via `docker-compose`.

As an illustration of the concept, we employ a Mininet-based network setup, depicted in Figure 6. The topology represents a single network domain comprising a controller and three data-plane switches. A Python script, developed using Scapy, transmits PCAP packets from the sending host to Switch 1. All three switches operate as P4 Behavioral Model version 2 (BMv2)² devices, utilizing the `v1Model` architecture. BMv2 acts as a reference software switch emulating `v1Model`-based programmable switches, thereby providing a virtual environment for developing, testing, and debugging P4 programs before deployment on physical hardware.

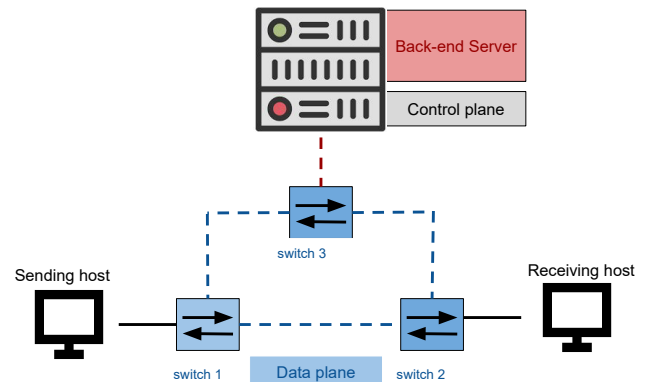


FIGURE 6. A simplistic topology in Mininet for proof of concept.

Switch 1 executes a `P4_16` program containing match-action rules that encode the trees of the deployed machine

¹PCAP2CSV.py in the SecFlow repository

²<http://bmv2.org/>

TABLE 2. Features derived and used to train the models. The symbol * indicates that the feature is included in training.

Feature Name	Description	Detector/Classifier			
		$D_{w,wo}$	$D_{w,wo}^T$	$D_{w,wo}^U$	$C_{w,wo}$
type	Binary value indicating the transport-layer protocol (TCP/UDP).	*,*	-, -	-, -	*,*
flow_bytes	Total bytes transmitted in either direction in a flow so far.	*,*	*,*	*,*	*,*
spkts	Number of packets transmitted by a source in a flow so far.	-,*	*,*	*,*	*,*
dpkts	Number of packets transmitted by a destination in a flow so far.	-,*	*,*	*,*	*,*
iat	Recent inter-packet duration in a flow, in seconds.	-, -	-, -	-, -	*, -
min_iat	Minimum inter-packet duration in a flow, in seconds.	-, -	*, -	*, -	*, -
max_iat	Maximum inter-packet duration in a flow, in seconds.	*, -	-, -	-, -	*, -
total_duration	Total duration of a flow up to the current packet, in seconds.	*, -	-, -	*, -	*, -
srate	Rate of packets transmitted from source to destination per second.	*, -	-, -	-, -	*, -
drate	Rate of packets transmitted from destination to source per second.	*, -	-, -	-, -	*, -
N_IN_Conn_Src_IP	Number of outgoing connections from the source IP.	*,*	*,*	*,*	*,*
N_IN_Conn_Dst_IP	Number of incoming connections to the destination IP.	*,*	*,*	*,*	*,*
avg_packet_len	Mean packet length in the current direction of a flow.	*,*	*,*	*,*	*,*
ttl	Time-to-live field value in the IP header.	*,*	*,*	*,*	*,*
state_rst	TCP packet with the reset flag set.	*,*	*,*	-, -	*,*
fin_flag	FIN flag in the TCP header.	*,*	*,*	-, -	*,*
syn_flag	SYN flag in the TCP header.	*,*	*,*	-, -	*,*
psh_flag	PUSH flag in the TCP header.	*,*	*,*	-, -	*,*
ack_flag	ACK flag in the TCP header.	*,*	*,*	-, -	*,*
state_con	TCP packet that is not the first packet in the flow and does not have RESET set.	*,*	-,*	-, -	*,*
state_int	First packet in a UDP flow.	*,*	-, -	-, -	*,*

TABLE 3. Match-action rule and table-stage footprint of the six detector variants, compiled for the Docker-based P4 testbed (Section IV-C). Features/Stages lists the feature count and the number of sequential match-action tables a packet may traverse for that detector; Rules is the total number of compiled table_add entries across all stages. ✓ indicates the traffic type each detector is trained on; × indicates it does not apply.

Detector	Features/ Stages	Rules	TCP	UDP	Time-based features
D_w	17/11	727	✓	✓	Yes
D_w^T	13/11	631	✓	×	Yes
D_w^U	9/9	203	×	✓	Yes
D_{wo}	15/11	551	✓	✓	No
D_{wo}^T	13/11	407	✓	×	No
D_{wo}^U	7/9	169	×	✓	No

learning (ML) model(s). When a packet is identified as *benign*, it is forwarded to the receiving host via the port connected to Switch 2. Conversely, packets classified as *malicious* are mirrored to the control plane through Switch 3. For simplicity, Switches 2 and 3 perform only forwarding functions without additional match-action logic. To improve scalability, every n^{th} packet within a malicious flow (e.g., every 3rd packet in Figure 4) is mirrored to the controller with a truncated

payload. This process continues until updated switch rules are applied to mitigate the flow, following controller attribution of its attack type, or the flow naturally terminates.

On Switch 1, the trained ML trees are translated into match-action rules similar to *SwitchTree* (21). Each node is represented as a tuple $\langle \text{node_id}, \text{feature_id}, \text{isTrue} \rangle$ (see Figure 7), where *node_id* denotes the node index, *feature_id* corresponds to the predictive feature, and *isTrue* indicates whether the feature value is less than or equal to a given threshold. For instance, rule 2 $\langle 16, 2, 1 \rangle$ specifies that at node 16, feature 2 is $\leq$ its threshold; the last parameter of the corresponding parent node (rule 1) stores the threshold value (i.e., 0 in $\langle 16, 2, 0 \rangle$). Nodes with three parameters at the end represent intermediate decisions, executed via the *CheckFeature* action to traverse subsequent nodes, whereas two-parameter nodes (e.g., rule 3 $\langle 18, 0 \rangle$) indicate leaf classifications handled by *SetClass*. Here, node 18 labels a packet as *benign* (class 0).

Unlike *SwitchTree*, *SecFlow* avoids hard-coded source and destination IPs to infer flow direction. Instead, it dynamically determines uplink and downlink flow orientation, ensuring correct feature extraction and thereby enhancing

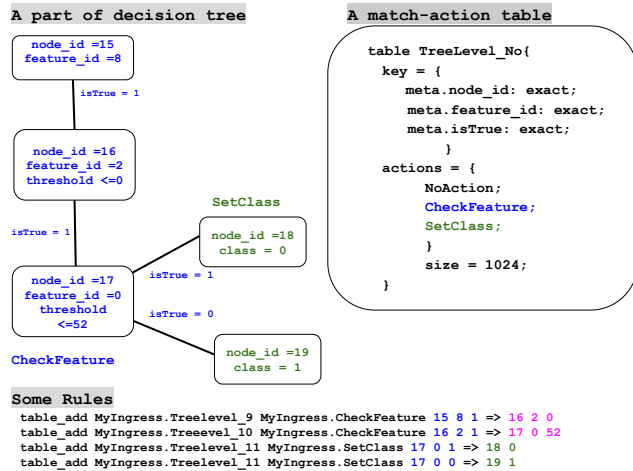


FIGURE 7. Part of a decision tree, with equivalent rules to be run by a match-action table.

```
number_attack_pkts[0]    = (737731 bytes, 2055
    packets)
counter_pkts[0]          = (797831 bytes, 2247
    packets)
reg_number_attack_flows[0]= 652
counter_flows[0]         = (232877 bytes, 685
    packets)
```

FIGURE 8. Status of malicious packets and flows recorded at the switch.

classification accuracy. The framework supports both flow- and packet-level detection in the data plane. The controller retrieves flow metadata and counters, as shown in Figure 8, to measure attack intensity. These counters and registers are refreshed whenever a new flow overlaps with an expired one after a timeout.

As mentioned earlier, every n^{th} packet per flow detected as malicious is mirrored to the controller for attribution. To minimize false positives, an IOption header is appended to the mirrored packets (as shown in Figure 9), encapsulating three key flow-level statistics: the `attack_packet` field, representing the number of malicious packets identified within the flow, and the `spkts` and `dpkts` fields, denoting the packet counts observed by the switch in the source-to-destination and destination-to-source directions, respectively. The controller periodically classifies flows where the majority of packets are flagged as malicious, thus reducing the influence of packet-level misclassifications, and computes the ratio of `attack_packet` to $(spkts + dpkts)$, rounding it to the nearest integer. The backend classifier is triggered upon receiving multiple alerts from the data plane; it processes only the most recent packet from a given flow, since, as noted in (40), labeling an entire flow as malicious based on a single packet yields high inaccuracy. Similar to (19), SecFlow considers a flow malicious only when multiple packets within it are flagged; specifically, when the computed ratio equals 1, the flow becomes a candidate for DDoS attack type classification; based on classification outcomes, the

```
// Encode flow-level statistics in the IOption
header
hdr.ipv4_option.pkts      = bit<16>(meta.dpkts
    + meta.spkts);
hdr.ipv4_option.attack_packet = bit<16>(meta.
    attack_packet);
```

FIGURE 9. P4 code snippet for sending flow information of malicious packets to the controller using an IOption header.

switch tables can be dynamically updated to block subsequent occurrences of similar attack flows.

Because Switch 1 executes the trained detector while Switch 2 and Switch 3 perform only stateless forwarding, per-process CPU and memory usage is sampled directly via `psutil`, polling each `simple_switch` process's resident set size (RSS) and CPU utilization at fixed intervals. Since all three switches share the same binary name, samples are logged concurrently and separated by RSS magnitude: a low group, attributed to Switch 2/Switch 3's stateless forwarding, and a markedly higher group, attributed to Switch 1's stateful detector; only the high-RSS samples are retained for the per-switch measurements reported in Section V.

For high-fidelity flow classification, we employ a Residual Network (ResNet) architecture, depicted in Figure 10. The model receives a single packet's feature vector, reshaped as $(L \times 1)$, where $L = 22$ for the Cw variant (all features) and $L = 20$ for the Cwo variant (excluding the two time-based features). The Conv1D layers learn local correlations across adjacent features in this vector, rather than across a temporal sequence of packets. The model leverages Conv1D layers for hierarchical feature extraction, augmented by residual (skip) connections that reintroduce earlier activations to mitigate vanishing gradients. The architecture begins with a convolutional layer (32 filters, 3×3 kernel), followed by max pooling for dimensionality reduction. The network then passes through $N = 4$ residual blocks arranged in two groups: the first group (Blocks 1–2) uses filter depths of 32 and 64, followed by max pooling; the second group (Blocks 3–4) uses filter depths of 64 and 128. Each residual block comprises two Conv1D layers, each followed by batch normalization and ReLU activation, with a projection shortcut applied when the filter depth changes. A global average pooling layer aggregates the learned feature maps, succeeded by a dense layer with 64 neurons (ReLU activation). Dropout regularization ($p = 0.5$) precedes the output layer, which employs a 9-neuron softmax for multiclass DDoS attack classification. The model uses the AdamW optimizer (learning rate = 0.001), categorical cross-entropy loss, and accuracy as its evaluation metric. Its configuration is summarized in Table 4, ensuring an efficient balance between complexity and performance for packet-level feature vectors.

To support reproducibility, the complete SecFlow codebase is publicly available³. The repository includes: the P4 programs implementing the data-plane detectors described in Sec-

³<https://github.com/mahfida/SecFlow>

TABLE 4. Architectural summary of the proposed ResNet-based classification model.

Layer / Block	Details	Activation	Output Dimensions
Input	Single packet feature vector: $L=22$ (Cw) or $L=20$ (Cwo), channels $C=1$	–	$(L, 1)$
Initial Conv1D	32 filters, kernel size = 3, stride = 1, padding = same	ReLU	$(L, 32)$
Max Pooling 1D	Pool size = 2	–	$(\lfloor L/2 \rfloor, 32)$
Residual Block 1	[Conv1D(32, k=3) → BN → ReLU] → [Conv1D(32, k=3) → BN], identity shortcut	ReLU	$(\lfloor L/2 \rfloor, 32)$
Residual Block 2	[Conv1D(64, k=3) → BN → ReLU] → [Conv1D(64, k=3) → BN], projection shortcut	ReLU	$(\lfloor L/2 \rfloor, 64)$
Max Pooling 1D	Pool size = 2	–	$(\lfloor L/4 \rfloor, 64)$
Residual Block 3	[Conv1D(64, k=3) → BN → ReLU] → [Conv1D(64, k=3) → BN], identity shortcut	ReLU	$(\lfloor L/4 \rfloor, 64)$
Residual Block 4	[Conv1D(128, k=3) → BN → ReLU] → [Conv1D(128, k=3) → BN], projection shortcut	ReLU	$(\lfloor L/4 \rfloor, 128)$
Global Average Pooling	Reduces spatial dimension to a single vector	–	$(128,)$
Dense Layer	64 neurons	ReLU	$(64,)$
Dropout	Dropout rate = 0.5	–	$(64,)$
Output Dense Layer	9 neurons (number of DDoS attack classes)	Softmax	$(9,)$

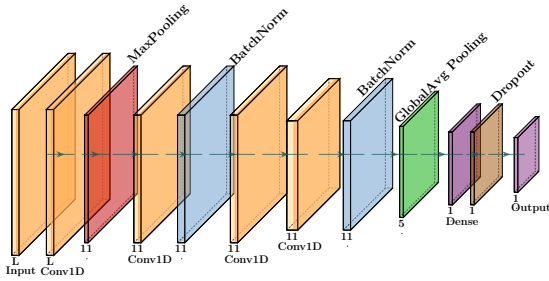


FIGURE 10. Our classification model architecture based on ResNet.

tion III; the preprocessing scripts (including PCAP2CSV.py) used to convert PCAP traces into the CSV feature sets described in Section IV-B; the training scripts and trained model weights for both the decision-tree detectors and the ResNet-based classifiers; and the exact train/test PCAP splits used to produce the results reported in Section V.

V. EVALUATION

For tree-based attack detection, we first investigate how the depth and number of trees in the Random Forest (RF) model affect the accuracy of malicious traffic detection. Five RF models, each with a fixed depth of 10, are tested using tree counts ranging from 1 to 5. The results, illustrated in Figure 11 (a, c), indicate that increasing the number of trees had minimal impact on accuracy improvement. Consequently, a simpler Decision Tree (DT) model, an RF with a single tree, is selected for further evaluation.

Subsequent analysis examines the influence of tree depth on the DT model. Depths of 5, 10, 15, 20, and 25 are assessed using 10-fold Cross Validation (CV). As shown in Figure 11 (b, d), a depth of 10 minimizes false positives across most detectors, while deeper models slightly enhance the

TABLE 5. Comparative packet- and flow-level F1-score, TPR, and TNR for TCP and UDP traffic, across the different detectors. All values are rounded to two decimal places.

Detector	Level	TCP			UDP		
		F1	TPR	TNR	F1	TPR	TNR
D_w	Packet	0.93	0.93	0.59	0.83	0.78	0.63
	Flow	0.99	0.98	0.88	0.55	0.71	0.83
D_{wo}	Packet	0.97	0.94	0.97	0.99	0.99	0.94
	Flow	0.99	0.99	0.86	0.63	0.86	0.84
$D_w^{T/U}$	Packet	0.96	0.94	0.94	0.98	0.97	0.95
	Flow	0.99	0.99	0.96	0.86	0.80	0.99
$D_{wo}^{T/U}$	Packet	0.97	0.95	0.96	0.99	0.99	0.99
	Flow	0.99	0.99	0.97	0.92	0.86	1.00

detection of malicious packets. Therefore, a depth of 10 is chosen as the optimal balance between detection accuracy and model complexity.

A. DETECTION OF MALICIOUS FLOWS

Based on the results presented in Figure 11, the attack detectors are trained using DTs with a depth of 10. These detectors are designed to identify packets associated with any of the malicious flows listed in Figure 1. The resulting models include D_w , D_w^T , and D_w^U that are trained on 17, 13, and 9 high-impact features respectively, while D_{wo} , D_{wo}^T , and D_{wo}^U are trained on 15, 13, and 7 features, with the latter group excluding time-based predictors. As discussed in Section IV, time-derived features computed live via `ingress_global_timestamp` can diverge from the PCAP-derived timestamps used at training time once a detector is deployed on the switch. We therefore evaluate all six variants directly on the Docker-based P4 testbed described in Section IV-C.

Table 5 reports packet- and flow-level F1-score, TPR, and TNR for all six variants, separated by TCP and UDP traffic. The most significant contrast is between the time-inclusive and

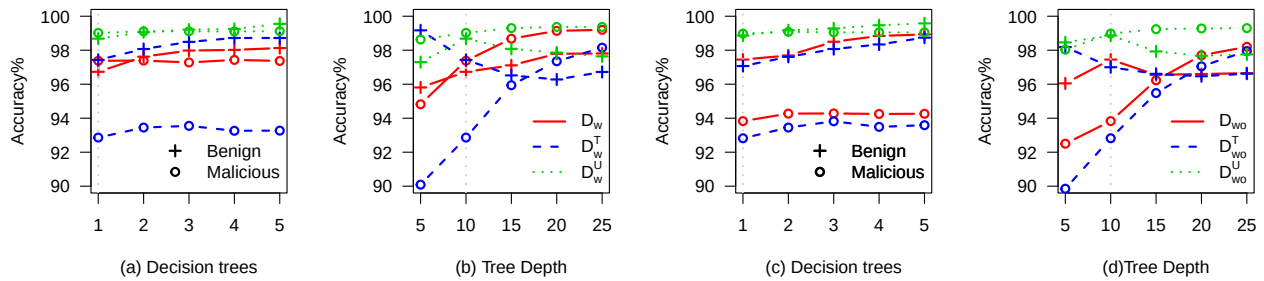


FIGURE 11. Training accuracy with different number of RF trees and Tree depths with (a, b) detectors including time-based features and (c, d) detectors excluding time-based features.

non-time-based families: D_w attains the lowest TNR of any detector on TCP traffic at the packet level, at 0.59, versus 0.94 or higher for every non-time-based detector, and the lowest F1-score of any detector on UDP traffic at the flow level, at 0.55, versus 0.86 or higher for every non-time-based detector. This is consistent with the timing-mismatch mechanism discussed in Section IV, since these detectors' thresholds are learned from PCAP-derived timestamps that no longer correspond to switch-measured inter-packet gaps once deployed.

Among the non-time-based detectors, $D_{wo}^{T/U}$ is the best or joint-best performer in every reported metric cell, and dominates every UDP metric outright at both the packet and flow level, reaching an F1-score of 0.99 against D_{wo} 's 0.99 at the packet level, where the underlying rates are close, and 0.92 against 0.63 at the flow level, a substantially wider margin. This asymmetry is consistent with the systematic differences between TCP and UDP traffic profiles noted in Section IV, since a single detector's thresholds are shared across both protocols and therefore represent a compromise fit, which costs UDP traffic more than TCP traffic given UDP's wider variation in packet size and volume. The sole exception is packet-level TNR on TCP traffic, where D_{wo} attains a marginally higher value, at 0.97 versus 0.96, a difference too small to affect the overall pattern.

A further pattern in Table 5 warrants explicit discussion. For UDP traffic, $D_{wo}^{T/U}$ attains a near-perfect packet-level TPR of 0.99, while its flow-level TPR is markedly lower, at 0.86. UDP flow-level rates are computed over a small number of test flows, 22 for UDP Flood and 13 for UDP Fragmentation as shown in Table 1, so this gap corresponds to only a small number of missed flows out of the 35 UDP attack flows in the test set, each requiring Section IV-C's strict criterion that nearly every packet in the flow be flagged for the flow itself to count as detected. The same small-sample sensitivity accounts for the corresponding, smaller gaps observed for the other detectors, and is a property of the test set and flow-level decision rule applied uniformly across all six detectors, rather than a weakness specific to $D_{wo}^{T/U}$.

Takeaways: D_{wo}^T and D_{wo}^U are recommended for deployment. Time-derived features cause a sharp accuracy collapse only in the combined detector D_w ; splitting by protocol alone already contains most of this damage, since $D_w^{T/U}$'s worst values

fall well short of D_w 's collapse, and excluding time-derived features removes the remaining gap, giving $D_{wo}^{T/U}$ the best or near-best result across every reported metric, with UDP traffic benefiting most. Section V-D shows this recommendation also holds on resource grounds, at no additional CPU or memory cost relative to the combined detector.

B. ATTRIBUTION OF FLOWS MARKED AS MALICIOUS

To enhance the accuracy of identifying whether a flow is indeed *malicious*, only the most recent mirrored packet of each flow is analyzed, utilizing its `IPOption` fields. Once the packet is confirmed as malicious, it is forwarded to the ResNet-based classification stage. Owing to the high computational efficiency of the ResNet architecture, a single classifier is employed for both TCP and UDP attack types. Moreover, to evaluate the impact of temporal information, two model variants are considered: one including time-based features (C_w) and another excluding them (C_{wo}). Like the data-plane detectors, C_w and C_{wo} classify packets that have already traversed the P4 switch, so switch-induced jitter in principle affects any timing feature derived from their arrival pattern regardless of where that feature is computed. The evaluation reported here trains and tests C_w/C_{wo} using features computed by the same Python-based pipeline directly from the PCAP files, under the assumption that the packet arrival rate at test time matches that of training, so that no differential jitter is introduced between the two phases. This is a narrower and more favorable condition than the one faced by the data-plane detectors, whose features are computed from live `ingress_global_timestamp` values under replay conditions that need not match training. Consequently, the strong performance of C_w reported here demonstrates that time-based features can be informative when arrival timing is consistent between training and inference, but does not establish that this consistency would hold if C_w were evaluated on mirrored packets subject to the same replay-rate variability documented for the data-plane detectors (Section V, Detection of Malicious Flows); validating C_w under such conditions remains an open question.

We evaluate and compare the accuracy of each trained classifier in detecting the three primary categories of DDoS attacks, as illustrated in Figure 1. Results are given in Table 6's

TABLE 6. Combined classification results for the C_w and C_{wo} models, organized by DDoS category. Per-class rows report precision, recall, and F1-score on the nine-class classification task; *Category Accuracy* rows report accuracy on a separate, coarser three-class classification task grouping all subtypes within that category.

Attack Class	C_w			C_{wo}		
	P	R	F1	P	R	F1
Volumetric						
TCP Flood	1.00	0.91	0.95	1.00	0.91	0.95
UDP Flood	1.00	1.00	1.00	1.00	1.00	1.00
Category Accuracy	96.5%			96.8%		
Protocol-based						
ACK Frag	0.71	0.96	0.82	0.92	0.89	0.91
PSH Flood	0.99	1.00	0.99	0.93	1.00	0.96
RSTFIN Flood	0.98	1.00	0.99	0.94	1.00	0.97
SYN Flood	0.99	0.85	0.91	0.91	0.60	0.73
UDP Frag	0.96	1.00	0.98	0.99	1.00	0.99
Category Accuracy	98.6%			96.1%		
Application-layer						
HTTP Flood	0.37	0.50	0.43	0.23	0.66	0.34
Slowloris	0.79	0.94	0.86	0.77	0.40	0.53
Category Accuracy	83.8%			75.4%		
Macro Avg	0.87	0.91	0.88	0.85	0.83	0.82
Weighted Avg	0.96	0.95	0.95	0.92	0.88	0.89
Overall Accuracy	0.95			0.88		

P: Precision, R: Recall, F1: F1-score.

Category Accuracy rows, reporting this separate, coarser three-class task. All models attain an accuracy exceeding 96% for volumetric and protocol-based attacks, with the time-inclusive classifier (C_w) reaching 98.6% on protocol-based attacks. Application-layer accuracy is comparatively lower for both, at 83.8% for C_w and 75.4% for C_{wo} .

Examining subcategory-level performance across all nine DDoS types (per-class precision, recall, and F1-score in Table 6; confusion matrices in Figure 12), both C_w and C_{wo} perform well overall, at 94.7% and 88.2% accuracy respectively, though C_w 's advantage does not extend to HTTP Flood, which both models confuse with Slowloris, SYN Flood, and ACK Frag.

C. ANALYSIS OF THE MISCLASSIFIED DDOS ATTACKS

The classification results in Figure 12 depict that HTTP Flood is a problematic class for both the C_w and C_{wo} models. The confusion matrices reveal that HTTP Flood attacks are frequently misclassified as Slowloris, suggesting that these two types of malicious traffic may have similar feature values. To identify the specific features responsible, we first apply permutation feature importance to highlight the features the model relies on most for prediction. This technique works by randomly shuffling each feature and observing how much the model's performance declines, with a large drop indicating high importance. The ten most important features are `flow_bytes`, `N_IN_Conn_P_Dst_IP`, `N_IN_Conn_P_Src_IP`, `avg_pkt_len`, `max_iat`, `iat`, `ack_flag`, `syn_flag`, `psh_flag` and `state_rst`. The presence of the time-derived features `max_iat` and

`iat` in this list is consistent with, not contrary to, their exclusion from the recommended data-plane detectors, for the reasons discussed in Section V (Attribution of Flows marked as Malicious). For the key features identified, we then compare the feature distributions of the *misclassified* HTTP Flood samples against those of the Slowloris samples and the *correctly classified* HTTP Flood samples to determine their role in misclassification, if any.

Figure 13(a) shows that `max_iat` feature has a very different distribution for *correctly classified* HTTP Flood samples compared to both *misclassified* HTTP Flood and Slowloris samples. To assess if the distributions of the features are significantly different, we use a non-parametric permutation test on the means of the corresponding distributions. With a significance level of $\alpha = 0.05$ and samples from *misclassified* and *correctly classified* HTTP Flood feature of `max_iat`, we get a $p - value = 0.0002$, indicating strong evidence that the two groups exhibit distinct patterns in this feature. The case is similar for the `avg_pkt_len` feature (see Figure 13 (b)), which likewise yields $p - value = 0$, further supporting the notion that the important flow-based features distinguish the *misclassified* HTTP Flood samples from their *correctly classified* counterparts.

In contrast, permutation test on `max_iat` and `avg_pkt_len` for *misclassified* HTTP Flood samples and the *correctly classified* Slowloris results in $p - value = 0.96$ and $p - value = 0.3882$, respectively, suggesting no meaningful differences between these groups. This lack of significance implies that the *misclassified* HTTP Flood samples share similar `max_iat` and `avg_pkt_len` characteristics with the Slowloris samples, which may contribute to the model's confusion between these two classes.

In operational terms, this confusion carries a practical cost: because HTTP Flood and Slowloris are both application-layer attacks but call for different mitigation responses (rate-limiting a high-volume HTTP flood versus terminating long-lived half-open connections for Slowloris), a misattributed flow risks the controller pushing the wrong mitigation rule – either under-mitigating a genuine flood or unnecessarily terminating legitimate long-lived connections misclassified as Slowloris. As a targeted mitigation, a promising direction is a secondary, binary confidence classifier trained specifically to distinguish HTTP Flood from Slowloris once the primary ResNet flags either class, rather than relying solely on the nine-way classifier's decision.

The confusion matrix in Figure 12 (a) shows that 14% of the tested HTTP Flood samples were misclassified as ACK Fragmentation attacks. To further investigate, we examined the distributions of key features and found that `flow_bytes` and `ack_flag` significantly contribute to this misclassification. Figure 13 (c) presents distribution of `flow_bytes`. The `flow_bytes` is almost static around 1514 bytes for ACK Fragmentation attacks with *misclassified* HTTP Flood samples closer to this value, compared to that of *correctly classified* HTTP Flood which tends to be larger in size. The analysis indicates a statistically significant difference in

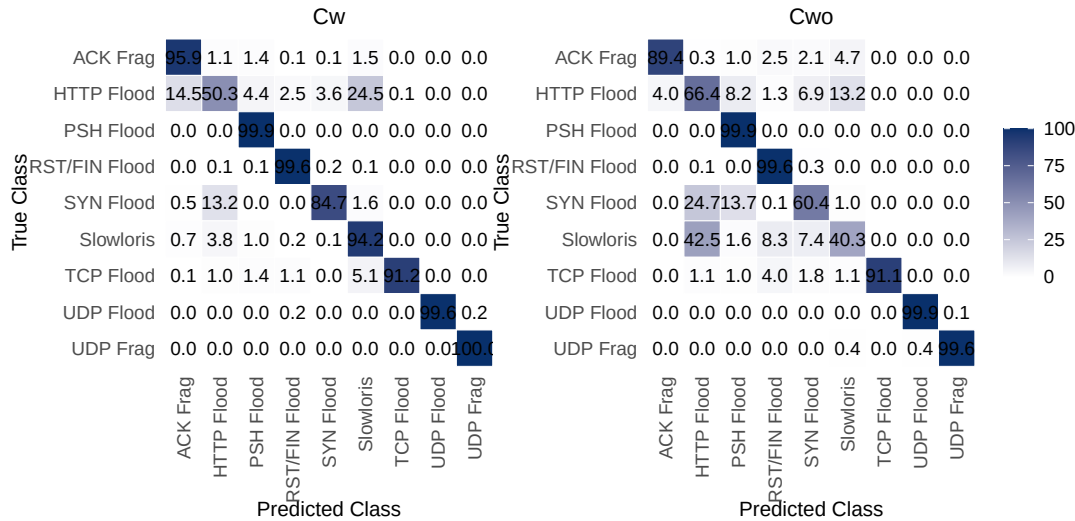


FIGURE 12. Comparison of classification accuracy using C_w and C_{wo} models on DDoS subcategories. Left: confusion matrix using C_w . Right: confusion matrix using C_{wo} .

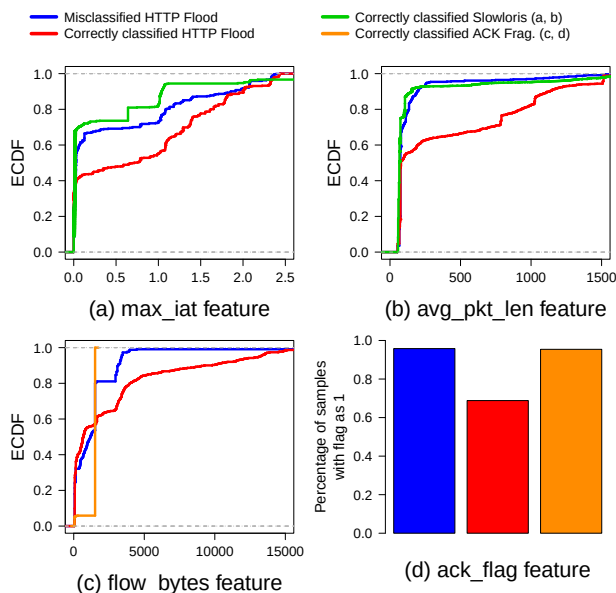


FIGURE 13. Features distributions for *misclassified* HTTP Flood, *correctly classified* HTTP Flood, and *correctly classified* Slowloris (a,b). Features distributions for *misclassified* HTTP Flood, *correctly classified* HTTP Flood, and *correctly classified* ACK Fragmentation (c,d).

the distribution of flow_bytes between the *misclassified* and *correctly classified* HTTP Flood samples ($p\text{-value} = 0$), suggesting that this feature is not deterministic for HTTP Flood. In contrast, the distribution of flow_bytes between the *misclassified* HTTP Flood samples and the *correctly classified* ACK Fragmentation samples shows no significant statistical difference ($p\text{-value} = 0.6069$).

Similarly, we observe that ack_flag is 1 around 95-98 % of the time for both *misclassified* HTTP Flood and ACK Fragmentation attacks (see Figure 13(d)), while it is

around 63% for that of *correctly classified* HTTP Flood samples. The permutation test shows a significant difference between the *misclassified* and *correctly classified* HTTP Flood samples ($p\text{-value} = 0.001$), while no significant difference is found between the *misclassified* HTTP Flood samples and the *correctly classified* ACK Fragmentation samples ($p\text{-value} = 0.8832$). This suggests that both flow_bytes and ack_flag play important roles in the model's confusion between HTTP Flood and ACK Fragmentation attacks.

D. IMPACT ON NETWORK PERFORMANCE

Resource measurements below use the Docker-based testbed described in Section IV-C, with per-switch CPU and memory isolated from background container noise via the RSS-threshold filtering described there. Bandwidth reduction is reported for three traffic conditions: benign traffic, TCP Flood, and UDP Flood. Delay, memory, and CPU cost are reported only for TCP Flood and UDP Flood, which serve as generalizing representatives of protocol-based/volumetric attack traffic on their respective transport protocols; benign traffic and the remaining seven attack subtypes are not separately profiled for these three metrics. The *Basic* switch baseline mirrors every received packet to the controller for inspection, using the same replay traffic, testbed, and measurement tooling as the *SecFlow* variants, isolating the comparison to the effect of in-network pre-filtering rather than differences in traffic or environment. For D_{wo}^T and D_{wo}^U/D_{wo}^U , whose topology places the classifier on Switch 1 alongside two plain-forwarding switches (Switch 2, Switch 3) running the same *basic.p4* program as the *Basic* baseline, the memory and CPU figures below reflect Switch 1 alone: readings matching the forwarding switches' own RSS range are excluded via the same RSS-threshold filtering described in Section IV-C, so that Switch 2/Switch 3's forwarding-only footprint is never

averaged into the reported classifier cost.

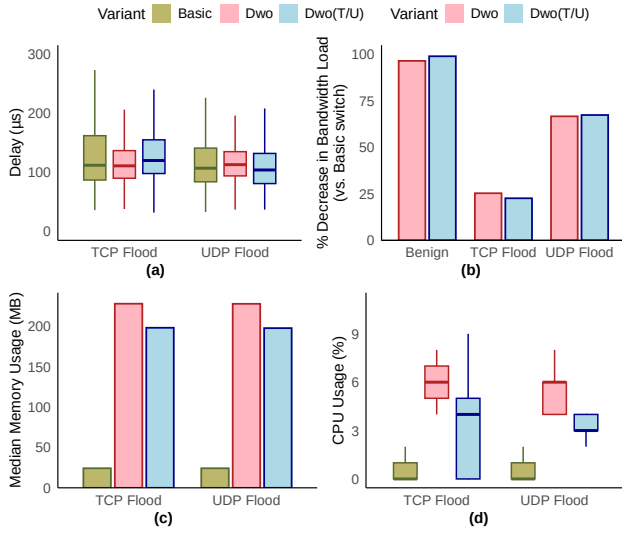


FIGURE 14. Delay, bandwidth reduction, memory usage, and CPU usage on the switch, comparing a *Basic* switch against D_{wo} and D_{wo}^T/D_{wo}^U . Panel (b) covers benign, TCP Flood, and UDP Flood traffic; panels (a), (c), and (d) cover TCP Flood and UDP Flood only, as generalizing representatives of the remaining attack subtypes on each transport protocol.

As shown in Figure 14(a), per-packet delay is broadly comparable across the *Basic*, D_{wo} , and D_{wo}^T/D_{wo}^U switch variants for TCP Flood and UDP Flood traffic, with medians in the 100–130 μs range for all three. This figure captures only the switch’s own packet-processing delay; it excludes the round trip to the controller, which the *Basic* switch requires for every packet in order to obtain a detection decision. Measured separately on the same testbed, this controller round trip has a median of 3.72 ms, one to two orders of magnitude larger than the in-switch delay in Figure 14(a). Because *SecFlow* performs detection directly on the switch, it avoids this round trip entirely for the detection decision itself, incurring it only for the subset of flagged packets subsequently mirrored for attack-type classification.

The classification models take approximately 1.3 ms to classify malicious packets into the three primary types of DDoS attacks and 1.6 ms to classify them into one of nine subcategories of DDoS attacks; this classification step is common to both architectures; and, unlike detection, it necessarily runs on the backend regardless of design.

Figure 14(b) shows that in-network detection substantially reduces the volume of traffic mirrored toward the controller relative to a *Basic* switch baseline that forwards every packet for inspection: bandwidth load toward the controller drops by approximately 97–99% for benign flows (which are never mirrored), 22–25% for TCP Flood, and 65–67% for UDP Flood, the larger reduction for UDP Flood reflecting its higher average packets-per-flow, since more of each flow’s traffic is suppressed once flagged.

Regarding resource cost (Figure 14(c, d)), evaluated on TCP Flood and UDP Flood traffic as generalizing cases, the *Basic*

switch, which uses no stateful registers or counters, consumes a median of approximately 24 MB of memory (up to 28 MB) and a median of 0% CPU (up to 5% for TCP Flood, 4% for UDP Flood). D_{wo} consumes substantially more: a median of approximately 228 MB of memory, consistent across both TCP Flood and UDP Flood, and a median CPU load of 6% (up to 11% for TCP Flood, 8% for UDP Flood). The protocol-specific D_{wo}^T/D_{wo}^U detectors use less of both: a median of approximately 198 MB of memory, and a median CPU load of 4% for TCP Flood (up to 9%) and 3% for UDP Flood (up to 6%). As noted above, these figures reflect Switch 1’s own resource consumption only, with Switch 2/Switch 3’s forwarding-only footprint excluded.

Takeaways: By performing detection directly on the switch, *SecFlow* avoids the 3.72 ms median controller round trip that a controller-based baseline requires for every packet, reducing detection latency by one to two orders of magnitude, while also cutting controller-bound bandwidth for malicious traffic by up to 67% for UDP Flood. This comes at a real resource cost relative to the *Basic* baseline, an approximately eightfold to ninefold increase in memory footprint (228 MB and 198 MB for D_{wo} and D_{wo}^T/D_{wo}^U respectively, against *Basic*’s 24 MB), isolated to the classifying switch alone. The protocol-specific detectors nonetheless incur this cost at lower overhead than the combined detector D_{wo} on every measured metric, confirming, on resource grounds, the same deployment recommendation reached in Section V (Detection of Malicious Flows) on accuracy grounds. Bandwidth reduction is demonstrated across three representative traffic conditions (benign, TCP Flood, UDP Flood), while delay, memory, and CPU cost are demonstrated on TCP Flood and UDP Flood as generalizing representatives of each protocol’s attack subtypes; resource behavior under benign traffic and the remaining seven attack subtypes individually has not been separately profiled.

E. COMPARISON WITH PRIOR IN-NETWORK SYSTEMS

To position *SecFlow* against representative prior in-network and hybrid DDoS detection systems, we benchmark it against *SwitchTree* (21), *pForest* (41), *IIsy* (31), and *BACKORDERS* (20) on a common Docker-based P4 testbed, using the same CICIOT2023 traffic categories described in Section IV. To keep the comparison fair, every system, including *SecFlow*, uses a single Decision Tree with a depth of 10 as its detection model, so differences in the results reflect each system’s architecture and feature design rather than an unequal choice of model complexity. Section VII details the rationale for selecting these four systems specifically.

As shown in Figure 15, every comparative system exhibits a severe accuracy collapse in at least one category, each in a different one: *IIsy* collapses on Benign traffic (0.26), indicating a high false-positive rate on legitimate flows; *pForest* collapses on TCP-A (0.17); *BACKORDERS* degrades sharply on TCP-A (0.39) and UDP (0.57); and *SwitchTree* drops on TCP-V/P (0.55). *SecFlow* is the only system that remains at or above 0.86 in every category, trading a narrower margin

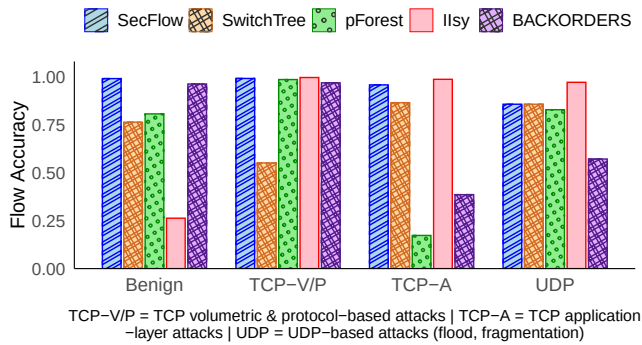


FIGURE 15. Flow-level accuracy of SecFlow and comparative methods across Benign, TCP volumetric/protocol-based (TCP-V/P), TCP application-layer (TCP-A), and UDP traffic categories.

against the single best performer in each category (e.g., IIsy on TCP-A and UDP) for consistency across the full attack taxonomy.

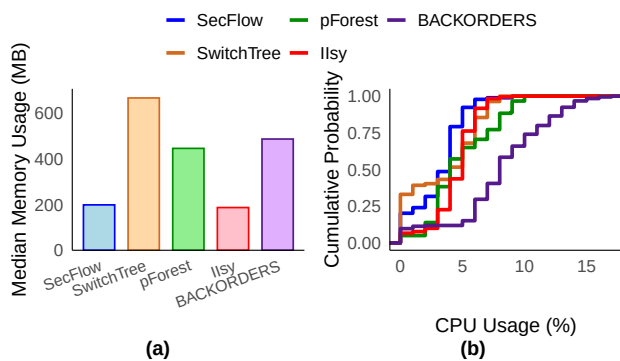


FIGURE 16. Median memory usage and CPU-usage cumulative distribution of SecFlow and comparative methods.

Figure 16(a) shows that IIsy has the lowest median memory footprint among the compared systems (approximately 186 MB), narrowly ahead of SecFlow at approximately 198 MB. pForest and BACKORDERS use substantially more memory, at approximately 444 MB and 485 MB respectively, and SwitchTree has the highest footprint of the five systems, at approximately 664 MB. Figure 16(b) shows that SecFlow and SwitchTree exhibit the fastest-converging CPU-usage distributions, with mean CPU usage across both flow types under 5%, while pForest and BACKORDERS converge more gradually, with BACKORDERS exhibiting the highest CPU usage of the five systems, with a mean of 6.9–9.4% and a median of up to 9% on UDP Flood.

Takeaways: Consistency, not category-by-category superiority, is what distinguishes SecFlow from the compared systems: it is the only evaluated system that avoids a severe accuracy collapse in any traffic category. On resource usage, SecFlow is competitive but not uniformly dominant, running a close second to IIsy on memory footprint while matching SwitchTree's low CPU usage; unlike its accuracy profile,

SecFlow's resource footprint does not stand out as the best of the five systems evaluated.

VI. DISCUSSION AND LIMITATIONS

The results in this paper should be read as a software-switch proof of concept rather than a validated hardware deployment: the accuracy, overhead, and comparative figures in Sections V-A–V-E were obtained on a Docker-based BMv2/Mininet testbed replayed at 24.79 pps, not on production hardware. Hardware switches such as Intel® Tofino™ operate at line rate with deterministic packet timing (17), which would likely improve the reliability of time-based features relative to our software testbed, but also impose architectural constraints, limited register memory, fixed pipeline depth, and restricted arithmetic, that may limit feature and model complexity in the data plane (31). Programmable switches additionally introduce their own attack surface: unverified P4 programs, compromised control channels, or adversarial rule updates can corrupt data-plane integrity (19), making code verification, runtime integrity monitoring, and strict data-/control-plane access control necessary companions to any production deployment (17).

On the question of whether a single, unified detector would suffice in place of SecFlow's dual-stage design, the combined, time-inclusive detector (D_w) is the least reliable of the six variants evaluated (Section V): its packet-level TNR on TCP traffic falls to 59%, and its flow-level F1-score on UDP traffic falls to 0.55, since timing-derived features no longer correspond to switch-measured inter-packet gaps once deployed. The recommended protocol-specific, non-time-based detectors (D_{wo}^T/D_{wo}^U) avoid this degradation, reaching a perfect flow-level TNR and 86% TPR on UDP traffic, demonstrating that a modular, protocol-specific pipeline is more robust than a single combined classifier. This robustness has a boundary, however: SecFlow's data-plane detector relies on a small set of 7–17 thresholded features, and an informed adversary could evade it by rate-shaping traffic below packet-count thresholds, padding packets toward the benign size distribution, or fragmenting a flow into many short sub-flows – limitations inherent to any threshold-based in-network detector, not unique to SecFlow, and each strategy necessarily reduces attack intensity while remaining potentially detectable by the backend ResNet's richer feature space. Hardening SecFlow against such evasion, via adaptive thresholds, feature randomization, or anomaly-based detection, and evaluating it under formal adversarial frameworks (42), remain open directions.

Several limitations otherwise inform future work. Results are derived entirely from CICIOT2023; the feature choices in Section IV (e.g., excluding IP-address-based identifiers) partially support generalization, but retraining would be needed for substantially different traffic, devices, or attack tooling, and validating the pipeline against broker-centric MQTT deployments remains a natural extension. SecFlow enables per-type mitigation rule updates but does not itself measure mitigation efficacy (e.g., time-to-block, false-block

rate), which is left to future work. HTTP Flood and Slowloris are frequently confused by the backend classifier (Section V); a secondary, binary confidence classifier distinguishing only these two classes once the primary ResNet flags either is a promising fix. Finally, determining the optimal number of packets per flow for reliable inference, exploring feature-computation methods within P4's arithmetic and memory limits, and assessing richer in-switch models (neural networks, K-means, ensembles) beyond the DT-based detectors used here are left for future work.

VII. RELATED WORK

Programmable data planes have been applied to a variety of network tasks such as congestion identification (44), in-network telemetry (45), load balancing (46), and traffic engineering (47). In the domain of DDoS detection, research splits into two main directions: (i) lightweight statistical/whitelisting methods, and (ii) machine-learning driven approaches. Statistical methods include heavy-hitter sketches (9; 10; 48), entropy-based anomaly detection (5; 6; 49), and static in-switch rules for traffic shaping or blocking (3; 4). These techniques are effective against volumetric floods but often fail against stealthy or multi-vector attacks.

The first wave of ML-driven systems highlighted the benefits and drawbacks of pushing intelligence into the data plane. (18) presented ORACLE in which switches export flow-level statistics as “report packets” to a controller, that employed Random Forest and KNN classifiers for DDoS traffic identification. However, classification delays at the controller prevented timely in-switch mitigation. Building on this, (41) proposed pForest that embedded Random Forest models directly into the switch pipeline, using a multi-phase design where each flow was progressively classified after a certain number of packets. While this reduced false positives, it assumed multi-packet flows, limiting its ability to detect single-packet or bursty DDoS attacks.

Lee et al. introduced SwitchTree, demonstrating the feasibility of embedding decision-tree ensembles directly inside a P4 switch (21). However, their implementation relied on hard-coded IP addresses to distinguish incoming from outgoing packets, making it tightly coupled to the UNSW-NB15 dataset (50). As a result, SwitchTree exhibits limited applicability and its detection capability is largely confined to volumetric DDoS attacks. (20) refined this idea, with Backorders by training an RF model on CICIDS2017 (51) and compiling its rules into P4 match-action tables. It focused on low-rate and slow DDoS attacks but did not distinguish attack types. (22) combined fast anomaly detection at the switch with detailed ML analysis at the controller. Named as Bungee-ML, the process reduced false negatives but only deciding whether an attack occurred rather than identifying its nature.

Recent work has explored richer models and resource-conscious deployment. (32) presented N3IC which embedded binarized neural networks (BNNs) into SmartNICs, showing throughput and latency advantages over CPU-based

classification, though limited to coarse binary decisions. (52), (43) proposed Mousika and Mousika-v2, respectively. These methods used distilled complex models (RF, GBDT, GRU, LSTM, MLP) into lightweight binary decision trees deployable in P4, balancing accuracy with switch resource constraints.

In mobile and edge contexts, (19) targeted 5G networks where IoT-driven DDoS floods (e.g., SYN, UDP, ICMP, DNS floods) are tunneled inside GTP-U headers. Their P4-based approach parsed GTP traffic to extract hidden inner IPs and exported statistics to an ONOS controller, where ML classifiers (logistic regression, Naïve Bayes) determined whether flows were benign or malicious. Mitigation was reactive, with new rules pushed to the P4 switch once multiple malicious verdicts were observed. While effective in a 5G testbed, detection was binary (attack vs. benign), and type classification was not addressed.

(17) proposed a high-speed IDS using Intel Tofino switches to export up to 31 flow features at 100 Gbps. The features were processed by a Random Forest classifier in the control plane. This reduced CPU load on Suricata but introduced dependency on export frequency and control channel capacity, which can become bottlenecks. In contrast, (16) introduced P4DDLe, which extracts packet-level features (including categorical values such as TCP flags and ICMP types) in the data plane and forwards them to a CNN-based IDS (called Lucid by (53)) in the controller. Evaluated on CIC-DDoS2019 (54), it successfully detected 13 distinct DDoS attack types, including reflection-based and exploitation-based floods, achieving lower false negatives than flow-only systems but at the cost of higher control-plane traffic.

Finally, (31) presented IISy, a hybrid in-network classification framework for mapping a wide range of ML models (Decision Tree, Random Forest, XGBoost, SVM, Naïve Bayes, K-means) onto off-the-shelf programmable switches. IISy runs a lightweight version of the model on the switch at line rate (6.4 Tbps) and defers uncertain cases to a backend for re-classification. This hybrid strategy reduces backend load by up to 70% while keeping latency sub-microsecond for most flows. It demonstrates near-optimal classification performance across diverse applications, including DDoS detection.

While the above works provide efficient detection, most either: (i) detect only a subset of attacks (e.g., SYN or slow DDoS) (19; 20); (ii) rely on control-plane classifiers with inherent delays (16; 17; 18); or (iii) classify traffic only as “attack” versus “benign” without identifying attack types (21; 22). By contrast, our proposed SecFlow introduces a dual-stage design: a lightweight, protocol-specific decision-tree detector at the data plane, coupled with a ResNet-based classifier at the backend server that attributes flagged flows to one of nine DDoS attack types with 95% weighted accuracy (macro-F1 0.88; Section V). This enables per-type mitigation strategies at a modest resource cost, while reducing link load toward the controller by up to 67% for malicious flows.

Table 7 surveys a broad range of prior systems, differing

TABLE 7. Comparison of representative DDoS detection approaches and SecFlow. Abbreviations: F = Flow features, P = Packet features, PF = Packet + Flow; D = Data plane, C = Control/Host; B = Binary (attack vs. benign), M = Multiclass (attack types).

Reference	Defense Against	Feature Scope	Inf. Loc.	Lbl.	Model	Dataset
ORACLE (18)	Specific DDoS	F	C	B	RF, KNN	CIC-IDS
pForest (41)	DoS/DDoS	F	D	B	RF (multi-phase)	CICIDS2017, UNIBS
SwitchTree (21)	Abnormal/Volumetric	PF	D	B	RF	UNSW-NB15
Backorders (20)	Slow/Low-rate DDoS	PF	D	B	RF → P4 rules	CICIDS2017
Bungee-ML (22)	Generic DDoS	PF	D + C	B	ML in controller	Mixed
N3IC (32)	DoS, Port-scan	F	D	B	BNN	UNSW-NB15
Mousika-v2 (43)	Flow/App/Malware	PF	D	B/M	BDT (distilled)	Various
Andrade et al. (19)	UDP/SYN/ICMP/DNS floods	F	C	B	LR, NB	CIC-2019
(17)	Multi-class Intrusions	F	C	M	RF	CIC-IDS
P4DDLe (16)	13 DDoS floods	P	D + C	B	CNN	CIC-DDoS2019
IIsy (31)	Generic DDoS + others	PF	D + C	B/M	DT, RF, XGB, SVM, NB, K-means	Multiple
SecFlow	9 DDoS Types	PF + sampled packets	D + C	M	DT (in-switch) + ResNet (server)	CICIoT2023

substantially in feature scope, inference location, label granularity, and dataset – differences that rule out a fully controlled comparison across the entire table. For the quantitative benchmarking in Section V-E, we therefore restrict the comparison to SwitchTree (21), pForest (41), IIsy (31), and BACKORDERS (20), which share three properties with SecFlow: (i) all perform *inline*, in-switch detection rather than relying on control-plane or off-path analysis; (ii) all target DDoS or closely related flooding traffic specifically, not generic intrusion detection; and (iii) all are, at their core, tree-based classifiers (Random Forest or Decision Tree), so observed differences in accuracy and resource usage are more attributable to architectural and feature-engineering choices than to a different modeling paradigm. N3IC (binarized neural networks), Mousika-v2 (distilled models), and P4DDLe (CNN-based, control-plane inference) were excluded on this basis, though they remain relevant qualitative points of reference in Table 7.

Beyond these shared properties, each baseline isolates a different design dimension from SecFlow. SwitchTree is the closest architectural predecessor, compiling a Random Forest directly into P4 rules via the same node-tuple encoding SecFlow uses (Section III); its original reliance on hard-coded source/destination IP roles, noted above, was replaced in this re-implementation with the same flow-generalizable uplink/downlink determination SecFlow uses, so the comparison isolates differences in feature selection and protocol-specific detector design rather than this dataset-coupling limitation. pForest’s multi-phase design, which classifies a flow only after accumulating several packets, isolates the effect of SecFlow’s single-packet detection, a difference expected to matter most for short or bursty flows. BACKORDERS targets low-rate and slow DDoS attacks, the same application-layer regime (HTTP Flood, Slowloris) where SecFlow’s own weakest results occur (Section V), making it a natural stress test for that category. IIsy is the only baseline that, like SecFlow, defers uncertain cases to a backend rather than deciding everything at the switch, isolating the effect of SecFlow’s dual-stage split. Section V-E details the shared

testbed, traffic, and model-complexity controls used in this comparison, and reports the results.

VIII. CONCLUSION

This paper introduces SecFlow, a dual-stage in-network framework that detects and attributes nine DDoS subtypes spanning volumetric, protocol-based, and application-layer attacks at line rate. Two lightweight, protocol-specific decision-tree detectors (D_{wo}^T for TCP, D_{wo}^U for UDP) achieve up to 99% flow-level accuracy on TCP flows and a perfect flow-level true-negative rate with 86% true-positive rate on UDP flows. Flagged flows are attributed by a backend ResNet classifier at 95% weighted accuracy (macro-F1 0.88), with per-class F1 scores mostly exceed 0.91 for volumetric and protocol-based attacks (with ACK Fragmentation the exception at 0.82), but fall to 0.43 (HTTP Flood) and 0.86 (Slowloris) for the two application-layer classes, reflecting feature overlap between them. By performing detection directly on the switch, SecFlow avoids the 3.72 ms median controller round trip a standard IDS requires for every packet, reducing detection latency by one to two orders of magnitude, while cutting controller-bound link load by up to 67% for malicious flows at a modest CPU (5–7%, up to 11%) and memory (under 230 MB) cost on the data plane.

Benchmarked against four representative in-network and hybrid DDoS detection systems (SwitchTree, pForest, IIsy, BACKORDERS) on a common testbed, SecFlow is the only system that avoids a severe accuracy collapse in any of four traffic categories, remaining at or above 0.86 throughout, though it is not uniformly dominant on resource usage, running a close second to IIsy on memory footprint; consistency across the full attack taxonomy, rather than superiority on every individual metric, is what distinguishes it from the compared baselines. Future work will validate SecFlow on hardware-programmable switches under live network conditions, evaluate its robustness against the adversarial evasion strategies discussed in Section VI, and assess its generalization beyond the CICIoT2023 dataset to broker-centric IoT deployments and other traffic profiles.

REFERENCES

- [1] O. Yoachimik, J. Pacheco, and Cloudforce One, "2025 Q4 DDoS Threat Report: A Record-Setting 31.4 Tbps Attack Caps a Year of Massive DDoS Assaults," Cloudflare Blog, 2026, accessed: 2026-09-08. [Online]. Available: <https://blog.cloudflare.com/ddos-threat-report-2025-q4/>
- [2] A. Pavlovic, "DDoS in 2025: The Year Automation Took the Wheel," Nokia Blog, Dec. 2025, accessed: 2026-09-08. [Online]. Available: <https://www.nokia.com/blog/ddos-in-2025-the-year-automation-took-the-wheel/>
- [3] B. Lewis, M. Broadbent, and N. Race, "P4ID: P4 enhanced intrusion detection," in 2019 IEEE conference on network function virtualization and software defined networks (NFV-SDN). IEEE, 2019, pp. 1–4.
- [4] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, "P4 Edge node enabling stateful traffic engineering and cyber security," *Journal of Optical Communications and Networking*, vol. 11, no. 1, pp. A84–A95, 2019.
- [5] R. N. Carvalho, J. L. Bordim, and E. A. P. Alchieri, "Entropy-based DoS attack identification in SDN," in 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2019, pp. 627–634.
- [6] D. Ding, M. Savi, and D. Siracusa, "Tracking normalized network traffic entropy to detect DDoS attacks in P4," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 4019–4031, 2021.
- [7] J. Huang, W. Zhang, Y. Li, L. Li, Z. Li, J. Ye, and J. Wang, "ChainSketch: An Efficient and Accurate Sketch for Heavy Flow Detection," *IEEE/ACM Transactions on Networking*, vol. 31, no. 2, pp. 738–753, 2023.
- [8] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*, 2017, pp. 164–176.
- [9] L. Tang, Q. Huang, and P. P. Lee, "A fast and compact invertible sketch for network-wide heavy flow detection," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2350–2363, 2020.
- [10] D. Ding, M. Savi, G. Antichi, and D. Siracusa, "An Incrementally-Deployable P4-Enabled Architecture for Network-Wide Heavy-Hitter Detection," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 75–88, 2020.
- [11] C. Ieracitano, A. Adeel, F. C. Morabito, and A. Hussain, "A novel statistical analysis and autoencoder driven intelligent intrusion detection approach," *Neurocomputing*, vol. 387, pp. 51–62, 2020.
- [12] M. A. Rahman, A. T. Asyhari, L. Leong, G. Satrya, M. Hai Tao, and M. Zolkipli, "Scalable machine learning-based intrusion detection system for IoT-enabled smart cities," *Sustainable Cities and Society*, vol. 61, p. 102324, 2020.
- [13] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed SDN controller system: A survey on design choice," *Computer Networks*, vol. 121, pp. 100–111, 2017. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1389128617301706>
- [14] N. M. Yungaicela-Naula, C. Vargas-Rosales, and J. A. Perez-Diaz, "SDN-Based Architecture for Transport and Application Layer DDoS Attack Detection by Using Machine and Deep Learning," *IEEE Access*, vol. 9, pp. 108 495–108 512, 2021.
- [15] B. B. Gupta and O. P. Badve, "Taxonomy of DoS and DDoS attacks and desirable defense mechanism in a cloud computing environment," *Neural Computing and Applications*, vol. 28, pp. 3655–3682, 2017.
- [16] R. Doriguzzi-Corin, L. A. D. Knob, L. Mendozzi, D. Siracusa, and M. Savi, "Introducing packet-level analysis in programmable data planes to advance network intrusion detection," *Computer Networks*, vol. 239, p. 110162, 2024.
- [17] N. Gray, K. Dietz, M. Seufert, and T. Hossfeld, "High performance network metadata extraction using P4 for ML-based intrusion detection systems," in 2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR). IEEE, 2021, pp. 1–7.
- [18] S. G. Macías, L. P. Gaspary, and J. F. Botero, "ORACLE: An Architecture for Collaboration of Data and Control Planes to Detect DDoS Attacks," in 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2021, pp. 962–967.
- [19] D. P. Andrade, K. Akkaya, A. Perez-Pons, S. Uluagac, and A. Sahin, "DDoS Attack Detection and Mitigation in 5G Networks using P4 and SDN," in 2024 IEEE 49th Conference on Local Computer Networks (LCN), 2024, pp. 1–9.
- [20] B. Coelho and A. Schaeffer-Filho, "BACKORDERS: using random forests to detect DDoS attacks in programmable data planes," in *Proceedings of the 5th International Workshop on P4 in Europe*, 2022, pp. 1–7.
- [21] J.-H. Lee and K. Singh, "Switchtree: in-network computing and traffic analyses with random forests," *Neural Computing and Applications*, pp. 1–12, 2020.
- [22] L. A. Quintero González, L. Castanheira, J. A. Marques, A. E. Schaeffer-Filho, and L. Paschoal Gaspary, "Bungee-ML: A Cross-Plane Approach for a Collaborative Defense Against DDoS Attacks," *Journal of Network and Systems Management*, vol. 31, no. 4, p. 77, 2023.
- [23] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese et al., "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.
- [24] C. Douligeris and A. Mitrokotsa, "DDoS attacks and defense mechanisms: a classification," in *Proceedings of the 3rd IEEE International Symposium on Signal*

- Processing and Information Technology (IEEE Cat. No. 03EX795). IEEE, 2003, pp. 190–193.
- [25] A. Praseed and P. S. Thilagam, “DDoS attacks at the application layer: Challenges and research perspectives for safeguarding web applications,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 661–685, 2018.
- [26] H. Kim and N. Feamster, “Improving network management with software defined networking,” *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.
- [27] M. Rahouti, K. Xiong, and Y. Xin, “Prototyping an sdn control framework for qos guarantees,” in *International Conference on Testbeds and Research Infrastructures*. Springer, 2020, pp. 3–16.
- [28] T. Jafarian, A. Ghaffari, A. Seyfollahi, and B. Arasteh, “Detecting and mitigating security anomalies in software-defined networking (sdn) using gradient-boosted trees and floodlight controller characteristics,” *Computer Standards Interfaces*, vol. 91, p. 103871, 2025. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0920548924000400>
- [29] M. Budiu and C. Dodd, “The p416 programming language,” *ACM SIGOPS Operating Systems Review*, vol. 51, no. 1, pp. 5–14, 2017.
- [30] D. D. Robin and J. I. Khan, “An Open-Source P416 Compiler Backend for Reconfigurable Match-Action Table Switches,” *arXiv preprint arXiv:2208.12892*, 2022.
- [31] C. Zheng, Z. Xiong, T. T. Bui, S. Kaupmees, R. Bensoussane, A. Bernabeu, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, “Iisy: Hybrid in-network classification using programmable switches,” *IEEE/ACM Transactions on Networking*, vol. 32, no. 3, pp. 2555–2570, 2024.
- [32] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, H. Haddadi, and R. Bifulco, “Re-architecting traffic analysis with neural network interface cards,” in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 513–533.
- [33] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [34] E. C. P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, and A. A. Ghorbani, “CICIoT2023: A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment,” *Sensors*, vol. 23, no. 13, 2023. [Online]. Available: <https://www.mdpi.com/1424-8220/23/13/5941>
- [35] K. Tools, “hping3 package description,” <https://www.kali.org/tools/hping3>, n.d., accessed 27-01-2024.
- [36] EPC-MSU, “UDP Flood,” <https://github.com/EPC-MSU/udp-flood>, 2020, accessed 28-06-2024.
- [37] Leon123, “Golang-HTTPFlood,” <https://github.com/Leon123/golang-httpflood>, 2019, accessed 28-06-2024.
- [38] G. Yaltirakli, “Slowloris,” <https://github.com/gkbrk/slowloris>, 2015, accessed 28-06-2024.
- [39] M. A. Salam, A. T. Azar, M. S. Elgendy, and K. M. Fouad, “The effect of different dimensionality reduction techniques on machine learning overfitting problem,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 4, pp. 641–655, 2021.
- [40] B. M. Xavier, R. Silva Guimarães, G. Comarela, and M. Martinello, “Map4: A pragmatic framework for in-network machine learning traffic classification,” *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4176–4188, 2022.
- [41] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever, “pforest: In-network inference with random forests,” *arXiv preprint arXiv:1909.05680*, 2019.
- [42] V.-T. Hoang, Y. A. Ergu, V.-L. Nguyen, and R.-G. Chang, “Security risks and countermeasures of adversarial attacks on ai-driven applications in 6g networks: A survey,” *Journal of Network and Computer Applications*, vol. 232, p. 104031, 2024.
- [43] G. Xie, Q. Li, G. Duan, J. Lin, Y. Dong, Y. Jiang, D. Zhao, and Y. Yang, “Empowering In-Network Classification in Programmable Switches by Binary Decision Tree and Knowledge Distillation,” *IEEE/ACM Transactions on Networking*, 2023.
- [44] M.-R. Fida, A. H. Ahmed, T. Dreibholz, A. F. Ocampo, A. Elmokashfi, and F. I. Michelinakis, “Bottleneck Identification in Cloudified Mobile Networks based on Distributed Telemetry,” *IEEE Transactions on Mobile Computing*, pp. 1–18, 2023.
- [45] D. Scano, A. Giorgetti, F. Paolucci, A. Sgambelluri, J. Chammanara, J. Rothman, M. Al-Bado, E. Marx, S. Ahearne, and F. Cugini, “Enabling P4 Network Telemetry in Edge Micro Data Centers With Kubernetes Orchestration,” *IEEE Access*, vol. 11, pp. 22 637–22 653, 2023.
- [46] J. Zhang, S. Wen, J. Zhang, H. Chai, T. Pan, T. Huang, L. Zhang, Y. Liu, and F. R. Yu, “Fast Switch-Based Load Balancer Considering Application Server States,” *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1391–1404, 2020.
- [47] S. Kamamura, “Dynamic Traffic Engineering Considering Service Grade in Integrated Service Network,” *IEEE Access*, vol. 10, pp. 79 021–79 028, 2022.
- [48] G. Cormode and S. Muthukrishnan, “An improved data stream summary: the count-min sketch and its applications,” *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.
- [49] L. D. Tsobdjou, S. Pierre, and A. Quintero, “An online entropy-based DDoS flooding attack detection system with dynamic threshold,” *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1679–1689, 2022.
- [50] N. Moustafa and J. Slay, “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)*, 2015, pp. 1–6.
- [51] I. Sharafaldin., A. Habibi Lashkari., and A. A. Ghorbani.,

- “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, INSTICC, SciTePress, 2018, pp. 108–116.
- [52] G. Xie, Q. Li, Y. Dong, G. Duan, Y. Jiang, and J. Duan, “Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation,” in IEEE INFOCOM 2022-IEEE Conference on Computer Communications. IEEE, 2022, pp. 1938–1947.
- [53] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martinez-del Rincon, and D. Siracusa, “Lucid: A practical, lightweight deep learning solution for ddos attack detection,” IEEE Transactions on Network and Service Management, vol. 17, no. 2, pp. 876–889, 2020.
- [54] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, “Developing realistic distributed denial of service (ddos) attack dataset and taxonomy,” in 2019 International Carnahan Conference on Security Technology (ICCST), 2019, pp. 1–8.



AZZA H. AHMED is an Associate Professor at Oslo Metropolitan University, Norway, and an Adjunct Research Scientist at the Simula Metropolitan Centre for Digital Engineering. She was previously a Postdoctoral Researcher at Simula Metropolitan, where she worked on 5G security, network automation, and in-network anomaly detection. She received her Ph.D. degree in Computer Science from Oslo Metropolitan University and Simula Metropolitan Centre for Digital Engineering, Norway, in 2023, with a focus on control principles for autonomous communication networks. She obtained her M.Sc. degree in Computer Science from the University of Nottingham, U.K., in 2012, and her B.Sc. degree in Electrical and Electronic Engineering (Communication Engineering) from the University of Khartoum, Sudan. Her research interests include programmable networks, network management and control, mobile broadband performance, Internet of Things (IoT) security, and the application of machine learning and artificial intelligence for network optimization and anomaly detection.



MAH-RUKH FIDA holds a PhD in Measurements and Analytics in Mobile Broadband (MBB) Computing from the University of Edinburgh, UK and Post Doc. in performance modeling in MBB networks from SimulaMet, Oslo Norway. She is currently a Senior Lecturer in the School of Business, Computing, and Social Sciences at the University of Gloucestershire, UK. In addition to her existing work on telemetry and performance evaluation within programmable networks, her research interests include applied data analytics and AI-based modeling, particularly in the fields of the Internet of Things and secure communications.



AMEER SHAKAYB ARSALAAN is a Lecturer in the Department of Computer Science at the University of Swabi, Pakistan. He completed PhD in 2024 at School of Computer and Mathematical Sciences, University of Adelaide, Australia, jointly funded by HEC Pakistan and University of Adelaide. Specializing in wireless networks, particularly MANETs, IoT, and QoI-based data transmissions, his work focuses on advanced routing protocols and adaptive algorithms to optimize network performance and fulfill user's requirements in decentralized systems. Recognized for his contributions to emergency response and autonomous vehicle networks, Arsalaan has published extensively and collaborated with industry and academic partners to bridge research with real-world applications in wireless communication.

...