USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

DDoS Detection at the Scale of One Hundred Tbps

Yunming Xiao, *The Chinese University of Hong Kong, Shenzhen, and Tencent;* Xijun Luo, Youliang Jiang, Aike Wang, Hu Chen, and Zhibin Zhou, *Tencent;* Heng Yu and Jiahao Cao, *Tsinghua University;* Yong Jiang, *Tsinghua Shenzhen International Graduate School;* Jilong Wang and Mingwei Xu, *Tsinghua University;* Yan Chen, *Northwestern University;* Congcong Miao, *Tencent*

https://www.usenix.org/conference/nsdi26/presentation/xiao

This paper is included in the Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation.

May 4–6, 2026 • Renton, WA, USA

ISBN 978-1-939133-54-0

Open access to the Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation is sponsored by

جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

DDoS Detection at the Scale of One Hundred Tbps

Yunming Xiao[1,2], Xijun Luo[2], Youliang Jiang[2], Aike Wang[2], Hu Chen[2], Zhibin Zhou[2],
Heng Yu[3], Jiahao Cao[3], Yong Jiang[4], Jilong Wang[3], Mingwei Xu[3], Yan Chen[5], Congcong Miao[2]
[1] *The Chinese University of Hong Kong, Shenzhen,* [2] *Tencent,* [3] *Tsinghua University,*
[4] *Tsinghua Shenzhen International Graduate School,* [5] *Northwestern University*

Abstract

Defending against Distributed Denial-of-Service (DDoS) attacks is a critical priority for cloud providers, who must manage ever-growing volumes of both benign and malicious traffic. While state-of-the-art DDoS detection systems leverage programmable devices to process traffic at hundreds of Gbps to Tbps on a single machine, large-scale cloud providers often handle traffic at scales approaching 100 Tbps. This two-orders-of-magnitude difference necessitates distributed processing across multiple servers, where new challenges are present. Specifically, naive load-balancing strategies lead to imbalanced traffic distribution and severe performance bottlenecks, while function offloading to programmable devices must balance flexibility and adaptability. In this paper, we present Canopy, a scalable DDoS detection system designed to overcome these challenges. Canopy features a dynamic load-balancing mechanism that adapts to fluctuating traffic patterns, ensuring balanced distribution across detection servers despite the mix of mice and elephant flows. Additionally, it employs a traffic compression technique at the programmable switch to significantly reduce per-server workload. These innovations enable Canopy to scale to over 100 Tbps in real-world deployments. Successfully deployed in production, Canopy has demonstrated its effectiveness in mitigating large-scale DDoS attacks.

1 Introduction

Distributed Denial-of-Service (DDoS) attacks have remained a persistent and evolving threat for decades. As the volume and sophistication of these attacks continue to escalate, defending against them has become a top priority for cloud providers [3]. Modern DDoS attacks employ a wide array of attack vectors, exploiting protocols such as ICMP, UDP, TCP, QUIC, and HTTP. Attackers further amplify their impact by leveraging reflection and amplification mechanisms inherent in protocols like DNS and SSDP, generating massive traffic volumes [50]. This diversity in attack strategies necessitates robust and adaptive defense mechanisms capable of addressing both the scale and complexity of these threats.

To mitigate the sheer traffic volumes associated with DDoS attacks, numerous solutions have been proposed. For example, research has shown that a single server equipped with FPGA-enabled SmartNICs can efficiently process 100 Gbps of network traffic [52]. Other approaches have pushed the limits further, achieving terabit-per-second (Tbps) line-rate processing using programmable switches [8, 34, 50].

While these advancements significantly boost single-machine processing capabilities, real-world data center traffic can peak at over 100 Tbps – two orders of magnitude beyond what a single server can handle. This motivates us to adopt a distributed processing approach, where coordinated efforts across server clusters are essential to safeguard entire data centers. However, it introduces two major challenges.

The first challenge in cluster-wide DDoS defense is determining how to efficiently and effectively distribute incoming packets across servers. Two primary approaches are commonly used, each with its own trade-offs. The first approach, a hash-based load balancer, assigns packets to servers based on header fields (e.g., source/destination IPs or ports). This ensures that all packets from the same flow are processed by a single server, eliminating the need for post-processing correlation and reducing latency. However, as traffic volume and processing complexity grow, individual servers may become overloaded, limiting scalability. The second approach, round-robin packet spraying, evenly distributes packets across all servers, preventing bottlenecks and improving load balancing. However, this method introduces additional overhead, as packets from the same flow must be reassembled and correlated, increasing analysis complexity, reaction time, and cache requirements. Consequently, neither approach is ideal for production networks, where traffic often follows a long-tailed distribution [6, 11] (§ 2.2).

The second challenge lies in leveraging programmable switches to boost system performance while maintaining flexibility and adaptability. While prior studies have demonstrated that a significant portion of DDoS detection functionalities

can be offloaded to programmable devices [50,52], real-world production environments demand greater flexibility and adaptability. For example, as a cloud provider, we operate a shared platform that allows users—both in-house development teams and third-party users—to define and update their own DDoS detection programs at any time. Offloading computation to programmable switches can be problematic in this context, as updating programs often requires reallocating resources and potentially rebooting devices, which is unacceptable in a production network (§ 2.3).

To address these challenges, we introduce Canopy, a robust DDoS defense system designed to mitigate attacks at scales of over 100 Tbps while balancing the competing demands of performance, flexibility, and adaptability.

For the first challenge, Canopy employs a dynamic load balancing strategy that intelligently alternates between hash-based load balancing and round-robin packet spraying based on real-time traffic conditions. Routing decisions are made dynamically by analyzing sampled packets over short intervals, enabling the system to adapt to changing traffic patterns. Since commodity switches lack the capability to execute such complex, real-time decision logic, Canopy leverages programmable switches as dynamic load balancers (§ 3.2).

For the second challenge, Canopy offloads functionalities to programmable switches with careful considerations of flexibility and adaptability. Specifically, we offload only packet parsing to the switches and introduce a traffic compression mechanism to reduce the processing load on downstream detection servers responsible for executing DDoS detection logic. This design enhances scalability by alleviating server-side computational demands while preserving the platform's ability to support any user-defined DDoS detection program, as detection remains on general-purpose CPUs (§ 3.3).

As a result, our evaluations show that Canopy's dynamic load balancing scheme efficiently manages traffic with long-tailed distributions, combining the strengths of hash-based and round-robin approaches. Meanwhile, the traffic compression mechanism enhances detection server throughput by an order of magnitude – from tens of Gbps to nearly 1 Tbps – while maintaining zero packet loss. Importantly, these improvements are achieved with minimal impact on the overall detection system (§ 5.1 and § 5.2).

To date, Canopy has been deployed in production data centers for over one year, operating smoothly while mitigating attacks across total traffic volumes reaching tens of Tbps, with peaks exceeding 100 Tbps – all without failure. Notably, it has demonstrated the capability to successfully handle traffic spikes that legacy systems failed to detect due to overwhelming load. Canopy maintains an end-to-end detection delay of approximately one second, primarily attributed to analysis at detection servers. For each month, Canopy successfully defended against hundreds of thousands of DDoS attacks, with attack bandwidths ranging from 100 Gbps to Tbps (§ 5.3).

This work **does not** raise any ethical issues.

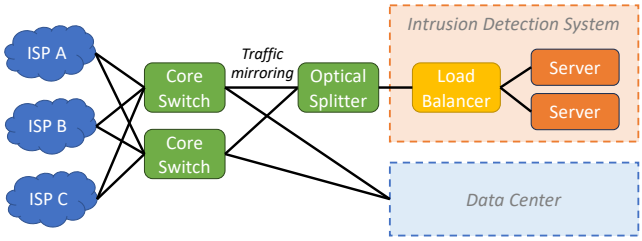


Figure 1: Architecture of DDoS detection system.

2 Background and Motivations

2.1 DDoS Attack Detection

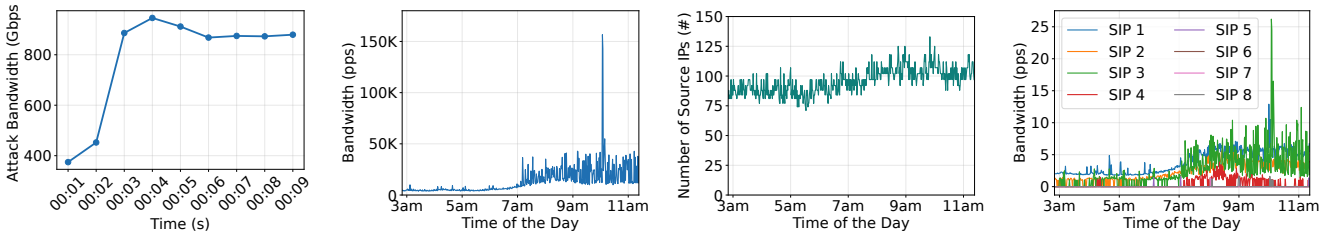
DDoS detection system procedure. DDoS attacks have grown increasingly prevalent, and as more companies migrate their infrastructure to cloud platforms, these platforms have become a primary target for such attacks. Consequently, DDoS detection and mitigation have become critical components of modern data center operations. A key requirement for any DDoS detection system is its ability to monitor and safeguard networks against malicious traffic while ensuring uninterrupted service for legitimate users. As illustrated in Figure 1, such systems typically operate by passively duplicating network traffic for analysis, minimizing disruption to the ongoing, primary data flow.

The process begins with Internet traffic arriving at core switches connected directly to ISPs. Optical splitters then duplicate the high-bandwidth optical signals. This passive duplication avoids the need to convert signals into electrical form, preserving the integrity and performance of the main traffic. The duplicated traffic is forwarded to security systems for analysis, while the primary traffic continues uninterrupted through the operational network.

The duplicated traffic is firstly routed through load balancers, which distribute it across a cluster of detection servers. Once the traffic reaches these servers, it undergoes in-depth inspection. When a potential DDoS attack is detected, the system generates alerts and activates mitigation strategies. These strategies may include traffic filtering, rate limiting, or blocking malicious sources. By combining real-time traffic analysis with advanced mitigation mechanisms, the system provides robust protection against a wide range of DDoS attacks and other emerging threats.

Operational insights. Our operational experience over the past few years reveals three key observations about evolving DDoS attacks. First, the scale of DDoS attacks is rapidly escalating. In our data centers, fewer than 500 attacks exceeded 100 Gbps in January 2023. By the end of that year, the number had doubled to more than 1,000, and within only the first half of 2024 it doubled again, recently surpassing 10,000 incidents per month (see § 5.3).

Second, attack traffic now ramps up to peak rates within just seconds. As shown in Figure 2(a), the traffic rate of a real-world DDoS attack surged to over 900 Gbps in only three



(a) An example of attack traffic.

(b) Total bandwidth; legitimate traffic.

(c) Number of source IPs; legitimate traffic.

(d) Bandwidth by source IP ; legitimate traffic.

Figure 2: Operational insight of DDoS traffic detection. (a) shows an example of the bandwidth of attack traffic towards one destination IP address (DIP), whereas (b)-(d) shows the bandwidth and number of source IPs of legitimate traffic to one DIP.

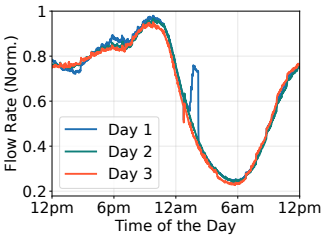
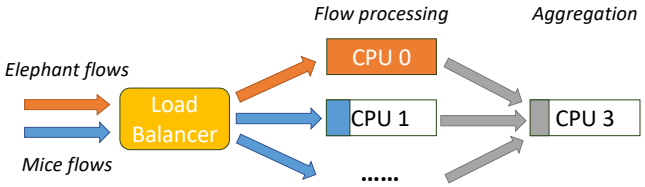
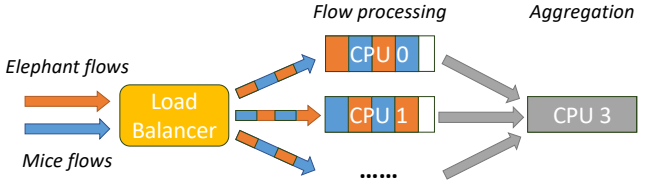


Figure 3: Daily traffic rate to our data centers.



(a) DDoS detection with hash-based load balancer.



(b) DDoS detection with round robin traffic sprayer.

Figure 4: Illustration of two load balancing schemes.

seconds, increasing by approximately 500 Gbps per second.

Third, DDoS attacks are becoming increasingly sophisticated. We observe the frequent emergence of sweep-segment attacks (which target one or more C-class subnets) and elephant-flow attacks. Moreover, adversaries increasingly craft traffic patterns that resemble legitimate workloads, which significantly complicates accurate detection. For instance, Figure 2(b) depicts the real-time packet volume received by a single destination IP. Sharp spikes occur around 10am. Yet, as shown in Figure 2(c) and 2(d), other indicators such as the number of distinct source IPs and their geographic distribution do not match typical DDoS signatures — in genuine DDoS incidents, the source IP count rises sharply and attacking botnets are usually geographically dispersed, rather than concentrated in a single region. Therefore, no DDoS attack happened in this case and single-egress threshold-based mechanisms would have produced false positives.

Traditional detection methods rely on rule-based systems and algorithms to identify suspicious patterns, such as unusual traffic spikes or protocol anomalies [7, 8, 20, 21, 34, 52]. More recently, machine learning-based detection algorithms have recently gained prominence in this field [12, 15, 24, 49], where advanced models trained on extensive datasets to detect subtle traffic patterns that might evade rule-based approaches.

2.2 Production Demand: Distributed Process

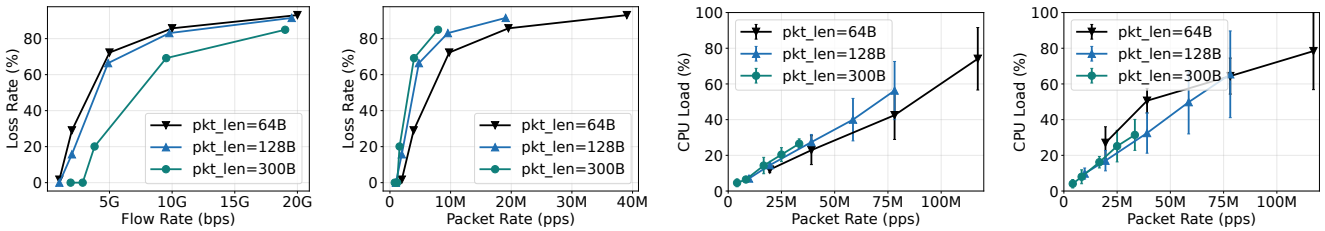
One major challenge in DDoS detection within modern data centers stems from the immense traffic volumes, which can reach over one hundred Tbps while characterized by long-tailed load distributions [6, 11]. Figure 3 shows the total flow rates to our data centers at different times of the day, ranging from tens of terabytes to peaks of over 100 Tbps. For confidentiality, specific values are omitted. Sudden traffic spikes occasionally occur – for example, on Day 1, a significant spike of tens of Tbps occurred around 2 am, driven primarily by a few elephant flows.

This scale far exceeds the capacity of the latest single-machine solutions, which typically handle 100 Gbps to 1 Tbps of traffic with the aid of SmartNICs or programmable switches [8, 34, 50, 52]. As a result, horizontal scaling – adding more servers to manage the incoming traffic – is essential. Nevertheless, in practice, we find that conventional load balancing schemes struggle to effectively handle long-tailed traffic distributions, particularly when dealing with the co-existence of elephant flows[1] (large, high-volume flows) and mice flows (smaller, low-volume flows), which have different impacts on server loads.

Hash-based load balance. By default, the load balancers depicted in Figure 1 adopt a hash-based distribution strategy. As shown in Figure 4(a), this strategy ensures that all packets destined for the same IP address are routed to the same server.[2] This design simplifies packet correlation and reduces

[1] Here, a flow is defined by a 2-tuple (src-IP, dst-IP), since packets traverse a layer-2 network between optical splitter and the load balancer (see Figure 1).

[2] Unlike load-balancing employed in data centers that usually hash by flow tuple, in DDoS detection it is advantageous to aggregate all packets



(a) Loss rate as function of flow rate; elephant flows; hash-based LB.

(b) Loss rate as function of packet rate; elephant flows; hash-based LB.

(c) CPU load as a function of packet rate; elephant flows; round-robin LB.

(d) CPU load as a function of packet rate; normal flows; round-robin LB.

Figure 5: Packet loss rate and CPU load when processing flows by a single server using different load balancing (LB) scheme.

processing latency. However, it also introduces a critical bottleneck when handling elephant flows. To illustrate this, we conducted experiments using a testbed equipped with a 48-core (196-thread) AMD EPYC processor. In this setup, 32 threads were dedicated to packet reception using DPDK, acting as 32 independent packet receivers, while the remaining threads were allocated for data analysis.

As demonstrated in Figure 5(a) and 5(b), the server quickly becomes overwhelmed under such conditions. For instance, with a packet length of 300B, a thread saturates its CPU utilization when traffic reaches 3 Gbps and begins experiencing a 20% packet loss rate when traffic reaches 3.8 Gbps or 153 Kpps (thousand packets per second). For smaller packets of 64B, the server struggles to handle even 1 Gbps of traffic without dropping packets. These limitations highlight the inefficiency of relying solely on hash-based distribution in high-volume scenarios with uneven traffic patterns.

Round-robin load balance. An alternative approach is round-robin packet spraying[3], where packets are evenly distributed across all servers, as illustrated in Figure 4(b). This method prevents any single server from being overwhelmed by high-traffic flows, offering improved load balancing and reliability, particularly for handling elephant flows. Indeed, experiments with the same setup as hash-based load balance confirm that no packet loss was found using round-robin spraying.

However, round-robin packet spraying introduces trade-offs. Packets from the same flow are spread across multiple servers, so results must be aggregated to provide a complete view of the network's DDoS status. This requires multi-level aggregation – within processes, across processes on a machine, and across machines. Aggregation keys, often IP addresses, pose challenges in large data centers, where unique IPs can number in the millions.

This duplication results in several inefficiencies. First, the duplication of keys increases resource usage and load, as each process must handle an inflated number of keys. Second, the inter-process aggregation step becomes a bottleneck due to the large number of redundant keys that need to be merged.

for a given destination IP at a single server/CPU core, as this improves data locality and avoids costly cross-server/core state access.

[3]Because we target volumetric DDoS attacks and operates on duplicated traffic rather than live traffic destined for production servers, packet reordering introduced by packet spraying is not a concern.

A small-scale experiment, shown in Figure 5(c) and 5(d), highlights this issue: normal flows impose a higher CPU load than a single elephant flow, and when packet lengths are 64B, some CPU cores reach 100% utilization at 120 Mpps.

Demands. Given these issues, there is a pressing need for a new load balancing design that is capable of effectively addressing both scenarios. Ideally, such a design would minimize the spraying of packets across multiple detection servers, thereby minimizing the detection latency and resource usage. At the same time, it should dynamically adapt to handle elephant flows by redistributing traffic across multiple servers when the capacity of a single server is exceeded.

2.3 Production Demand: Flexibility

After load balancing, the next step is the data analysis stage, where another primary challenge arises: balancing performance and flexibility. State-of-the-art DDoS detection solutions often rely on programmable devices, such as programmable switches or SmartNICs [8, 34, 50, 52], to enhance single-server processing capabilities by offloading specific functions to these devices. However, this approach comes with a notable trade-off: reduced flexibility and adaptivity. For example, programmable devices may not support certain functions/operands, or updating the DDoS detection algorithm may incur additional costs, such as requiring temporary service downtime when changes involve these devices.

The degree of offloading depends on the flexibility requirements. Systems like Peregrine [8] offload packet processing and feature computation, while others, such as Poseidon [50] and Pigasus [52], aim to offload as much functionality as possible, within the constraints of the hardware.

Our platform differs from these studies in that it is driven by real-world production requirements and must address practical challenges rather than serving purely as a research exploration. Specifically, our platform is designed to support traffic detection across an entire data center, where high flexibility and availability are critical necessities. Given the unpredictable nature of future demands and the evolution of DDoS detection algorithms, our design must enable most changes – such as the implementation of a new DDoS detection algorithm – to be executed swiftly and seamlessly without causing service disruptions or requiring system downtime.

In general, designing such a system requires careful consideration of what to offload and what to keep on the server. We argue that there is no universally "correct" design; rather, the optimal approach depends on the specific requirements and constraints of the use case.

2.4 Threat Model

We focus on volumetric DDoS attacks targeting victim destinations within data centers. In this scenario, we assume that attackers operate from outside the data center and that none of the components in our intrusion detection system are compromised. The traffic mirrored to our intrusion detection system includes both benign and malicious traffic. Attackers operate within a fixed budget to acquire or rent a large number of bots in a botnet with the aim of overwhelming the victim's bandwidth or computational resources. They can employ a combination of various DDoS attack types (e.g., Smurf attacks, SYN floods, ICMP/UDP floods, elephant flow attacks, DNS reflection attacks, NTP amplification attacks, HTTP floods, Slowloris attacks, etc.) and launch multiple attacks simultaneously. Additionally, attackers can dynamically modify the types/composition of attacks during operation.

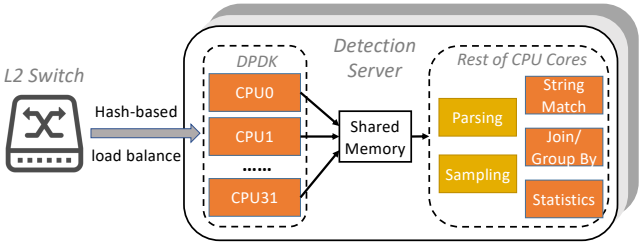
3 System Design

In this section, we first present a high-level overview of our system proposal (§ 3.1), before diving into the key designs, superficially a dynamic load balancing scheme (§ 3.2) and optimizations for the packet processing (§ 3.3).

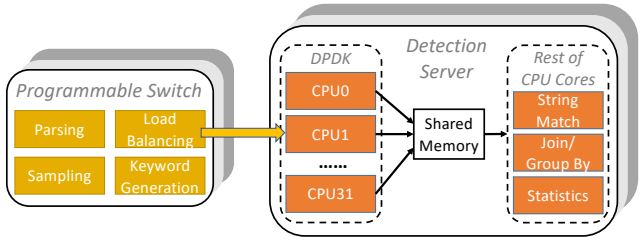
3.1 High-Level Overview

We propose Canopy, a system that integrates a programmable switch with downstream detection servers to efficiently handle high-throughput network traffic. The programmable switch is responsible for pre-processing network packets, extracting and compressing traffic features (e.g., DDoS-related keywords), dynamic load balancing, and forwarding these compressed features to detection servers for in-depth analysis and correlation. Figure 6 illustrates the system architecture and its end-to-end workflow, comparing it to the legacy system.

When a packet arrives at the programmable switch, it is first processed by the Parser module, which analyzes packet headers. The Parser supports a wide range of packet types, including IPv4, IPv6, tunnel packets (encapsulated with IPv4, IPv6, or GRE), and protocols such as TCP, UDP, and ICMP. After parsing, the switch applies random sampling to select a subset of packets for further processing. For each sampled packet, the switch extracts key feature information, such as flow identifiers and protocol-specific attributes. These features are stored in the switch's Registers until a sufficient volume of data is accumulated. Once enough features are



(a) Legacy system.



(b) Canopy.

Figure 6: Overview of Canopy and legacy detection system.

gathered, the switch consolidates, compresses, and encapsulates this information into the payload of a custom Ethernet packet. This compression mechanism significantly reduces the volume of data sent to the servers, optimizing bandwidth usage and computational resources (§ 3.3). By offloading packet parsing, feature extraction, and compression to the programmable switch, Canopy alleviates the computational burden on the detection servers, enabling them to focus on advanced analysis tasks that require greater flexibility and computational power.

A key feature of Canopy is its dynamic load balancing mechanism at the programmable switch (§ 3.2). The switch intelligently distributes traffic across downstream servers based on real-time load conditions. This is achieved by monitoring the bandwidth of each flow and adjusting the distribution of Ethernet packets accordingly. Importantly, the load balancing is performed per CPU thread, as each CPU thread dedicated to DPDK on the servers acts as an independent receiver.

The detection servers, equipped with high-performance frameworks like DPDK, receive the compressed packets and perform advanced processing. A subset of CPU cores is dedicated to DPDK for receiving Ethernet packets and caching them into an I/O queue, which is shared with subsequent analysis processes. The remaining CPU cores handle tasks such as field selection, flow correlation, rule-based decision-making, and statistics aggregation, enabling comprehensive traffic analysis, anomaly detection, and DDoS mitigation.

The combination of a programmable switch and commodity servers provides a flexible and scalable architecture. The switch handles high-speed, fixed-function tasks, while the servers offer the computational flexibility needed to adapt to evolving DDoS detection algorithms and traffic patterns. Random sampling ensures that the system can handle high-throughput traffic without overwhelming the detection servers,

while packet compression minimizes bandwidth usage and reduces the frequency of server-side processing. Overall, Canopy is a cost-effective and future-proof solution and is able to scale to meet the demands of modern data centers.

3.2 Dynamic Load Balance

As discussed in § 2.2, neither hash-based nor round-robin load balancing strategies are suitable for DDoS detection systems where traffic volumes exhibit long-tailed distributions. Other common load-balancing strategies, such as those based on Least Response Time or Least Connections, also fail to address the unique requirements of our scenario. The primary reason is that these strategies either require communication between load balancers (e.g., the programmable switches depicted in Figure 6) or depend on real-time feedback from the servers to make routing decisions. In our case, such feedback mechanisms are impractical due to the need to maximize throughput and minimize latency.

Our observation is that hash-based load balancing performs effectively in scenarios dominated by mice flows, while round-robin load balancing is better suited for environments with elephant flows. Based on this insight, we propose a dynamic load balancing design that achieves the best of both worlds – reaching a balance between evenly distributing load based on IP addresses and mitigating the impact of high load caused by elephant flows.

Specifically, our approach dynamically monitors traffic patterns in real time to adapt to changing conditions. Over a constant interval, we track the occurrence frequency of all observed IP addresses. If the number of packets associated with a particular IP exceeds a predefined elephant flow threshold, we distribute packets from this IP using round-robin spraying across multiple servers. For IP addresses below the threshold, we retain hash-based distribution, ensuring efficient correlation of packets belonging to the same flow.

Formally, let T be the threshold, M be the number of downstream detection servers, and $C(ip)$ be the function that counts the number of packets per IP address, the dynamic load balancing *DynoHash* works as follows:

$$DynoHash(ip) = \begin{cases} Hash(ip) \bmod M, & C(ip) \leq T \\ (Hash(ip) + C(IP)) \bmod M, & otherwise \end{cases} \quad (1)$$

By combining real-time monitoring with adaptive routing strategies, our dynamic load balancing design addresses the shortcomings of traditional methods and is well-suited for large-scale DDoS detection systems operating in high-performance environments.

It is noteworthy that this load balancing scheme is not limited to our system but applies to other DDoS detection systems as well. In a broader context, the processing capability of a single machine will always face inherent limitations, regardless of advancements in underlying hardware. As network traffic volumes continue to grow, scenarios where a single machine becomes a bottleneck – particularly when processing elephant flows – are inevitable. Therefore, load balancing remains highly relevant even in evolving production setups. While advancements in hardware and infrastructure may shift the balance points in the trade-offs, the fundamental need for effective load distribution will persist as a critical aspect of scalable and efficient DDoS detection.

```
struct ddos_keyword{
    bit<64> flow_label;
    bit<64> src_ip_high;
    bit<64> src_ip_low;
    bit<64> dst_ip_high;
    bit<64> dst_ip_low;

    bit<16> raw_len;
    bit<16> ip_len;
    bit<16> data_len;
    bit<16> dport;
    bit<16> sport;

    bit<8> tunnel_protocol;
    bit<8> protocol;
    bit<8> key_flag_a;
    bit<8> key_flag_b;
    bit<8> type_tcphl;
    bit<8> code_tcpflag;
    bit<32> tcp_ack;
    bit<32> tcp_seq;
}
```

Figure 7: Predefined keywords for DDoS detection.

3.3 Packet Processing Optimizations

3.3.1 Debate on function offloading

Compared to SmartNICs, programmable switches are better suited to achieve dynamic load balance introduced in § 3.2, as load balance extends beyond a single server. As programmable switches are adopted, they also present an opportunity to reduce workloads in the subsequent DDoS detection stage. Prior research has demonstrated the potential of programmable switches for various optimizations. For example, Poseidon [50] showcases offloading specific functions to programmable switches, while Jaqen [34] and Peregrine [8] highlight their effectiveness in implementing sketches that facilitate downstream processing.

However, existing solutions lack the flexibility required for our use case. As a cloud provider, we must support a platform capable of accommodating any DDoS detection algorithms provided by users – whether developed in-house or by third-party services deployed in our data centers. To address this, we have developed a platform that provides a set of traffic features and a dedicated programming language for DDoS detection, enabling users to program their detection logic and update it dynamically. Offloading computation to programmable switches is unsuitable for this purpose, as each update would require a new round of resource optimiza-

tions [34, 50], potentially involving pauses or reboots of the programmable switch, which is unacceptable in production.

Instead, we leverage programmable switches solely for packet parsing and traffic compression, significantly reducing the workload on detection servers. This approach is grounded in two key observations:

- Processing raw packet headers and payloads imposes a substantial computational burden on servers, as they must extract meaningful features from the data. This not only increases processing time but also delays detection. Programmable switches, with their line-rate processing capabilities, are far more efficient at extracting and formatting key information from packet headers.
- While DDoS detection algorithms evolve rapidly, the underlying network protocols – particularly those governing Internet packets – remain relatively stable. This stability arises from the need for consensus among multiple stakeholders (e.g., users, servers, middleboxes) to ensure successful packet transmission. As a result, it is feasible to predefine a set of keywords for DDoS detection without frequent updates.

Based on these insights, we propose to compress traffic: using programmable switches to extract, format, and compress traffic data before forwarding it to downstream servers.

3.3.2 Traffic compression at programmable switch

This subsection describes how Canopy performs traffic aggregation on the programmable switch. The design is nonetheless not straightforward because of hardware constraints: it lacks native support for multi-packet operations and restricts each register to be accessed at most once per packet. We first explain the selection of DDoS keywords and packet truncation, then describe how packet assembly is realized using registers and recirculation, and finally analyze the resulting resource consumption of our approach.

① DDoS keywords and ② packet truncation. We define a custom header structure specifically tailored to the DDoS detection scenario, which covers critical fields in IP, TCP/UDP, tunneling protocols, as illustrated in Figure 7. We select these fields based on our operational experience, and they cover all the DDoS detection algorithms we have deployed.[4] The keywords header fields are populated by both the packet `parser` and later match-action stages in the Ingress pipeline. After that, the header is forwarded to the `deparser` and then the Egress pipeline. At this stage, the packet consists of three parts: an Ethernet header, DDoS keywords, and the remaining payload which is useless for DDoS detection.

To optimize performance, packets should be truncated, *i.e.,* removing the unprocessed payload, and retaining only DDoS keywords. However, Tofino hardware does not natively support direct truncation of packets in the middle of the pipeline. The only component capable of truncating packets is leveraging the `mirror` module, which allows retaining a fixed number of bytes from the start of the packet. In Tofino hardware, there are two places where `mirror` is allowed to execute: "Ingress-to-Egress" and "Egress-to-Egress". The former mirrors the original packet entering the Ingress pipeline and puts it at the beginning of the Egress pipeline. Unfortunately, this means that it cannot help to truncate the processed packets after the Ingress pipeline with the DDoS keywords, and does not satisfy our demands. The latter mirrors the packet after it has been processed by the Ingress pipeline and puts it at the beginning of the Egress pipeline. We thus use the Egress-to-Egress `mirror` for packet truncation. Specifically, we mirror the first 78 bytes of the packet, including the Ethernet header and all DDoS keywords, and drop the original packet.

③ Packet Assembly. The above processing produces truncated packets containing only an Ethernet header and DDoS keywords for a single packet, reducing the workload on the downstream detection server. However, the high packet-per-second (PPS) rate may still overwhelm the detection servers. To mitigate this, we implement a packet assembly mechanism within the programmable switch pipeline, combining DDoS keywords from multiple packets into the payload of a single Ethernet packet. This approach significantly reduces the number of packets sent to the servers, lowering the PPS rate and enhancing system efficiency.

Implementing packet assembly on a programmable switch is challenging, as the switch processes packets individually and lacks native support for multi-packet operations. To overcome this, we devised a mechanism to temporarily store data from previous packets and embed the extracted content into the final packet. This solution, however, must address two key constraints: limited register space and restricted register access per stage at programmable switch.

The first challenge arises from the fact that the only component in the programmable switch capable of caching data is the `register`, which has a limited bit-width of 32 bits. This limitation makes it difficult to directly store the full 64-byte DDoS keyword (see Figure 7). To overcome this, we split each 64-byte DDoS keyword into 16 segments, each stored in a separate register object. Each register is allocated an address space of 256, enabling parallel storage of multiple segments of the same header field. The register structure is depicted in Figure 9. During packet processing, these segments are written to the registers in parallel, allowing efficient caching and retrieval of complete DDoS keywords.

The second challenge stems from the limitation that a register can perform only one read or write operation per address during a single packet processing cycle. Simultaneous reads, writes, or a combination of both to the same address are not permitted. To address this, we implement a loopback-based mechanism for assembling DDoS keywords across multi-

[4] For confidentiality reasons, unfortunately, we do not disclose the detection algorithms, so as to prevent potential exploitation of our data centers.

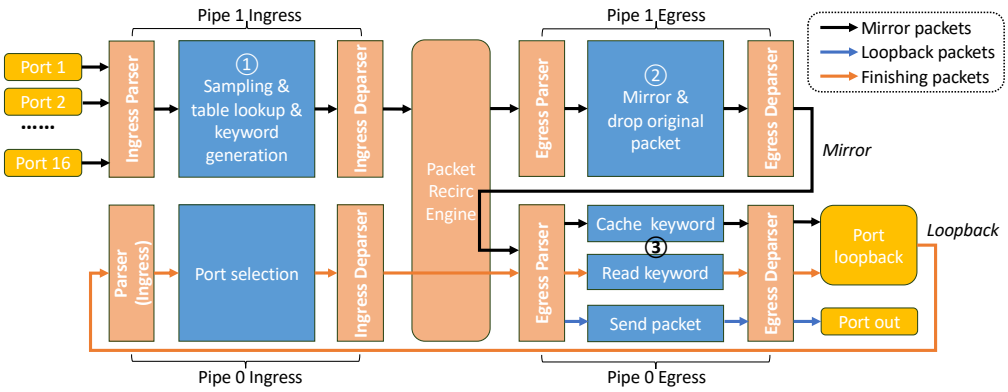


Figure 8: Programmable switch pipeline design.

Register 0 flow_label[63:32]	Register 1 flow_label[31:0]	Register 2 src_ip_high[63:32]	.....	Register 15 tcp_seq[31:0]
Addr0	Addr0	Addr0		Addr0
Addr1	Addr1	Addr1		Addr1
Addr2	Addr2	Addr2		Addr2
.....	.....	.....		.....
Addr255	Addr255	Addr255		Addr255

Figure 9: Registers used for caching DDoS keywords on the programmable switch. Sixteen 32-bit registers are allocated to hold a single complete DDoS keyword in Figure 7.

ple packets. Each mirrored packet contains a set of DDoS keyword segments, which are stored in different registers. This process is coordinated by two auxiliary variables: (i) `write_ptr` and (ii) `read_addr`. The first variable represents a register that tracks the next address for storing new DDoS keyword segments, incrementing with every processed packet. The second variable is a local variable that maintains the current address for reading data from the registers during keyword assembly.

Once DDoS keywords from N packets have been cached, the system initiates the loopback process (a.k.a. recirculation) to assemble them. Due to the pipeline's restriction of one register access per packet processing cycle, reading N keywords requires N loopback/recirculation packets. Each loopback packet retrieves a set of DDoS keyword segments from the registers, which are then appended to the payload by the Deparser. This process repeats until all N keywords are aggregated into a single packet payload. A termination flag in the final loopback packet signals the end of the aggregation, and an Ethernet header is appended before transmitting the aggregated packet to downstream servers. This workflow is illustrated in Figure 8.

Resource consumption. The above design effectively compresses traffic, significantly reducing the PPS rate to downstream servers while ensuring that the assembled packets contain all necessary DDoS keys. However, it also introduces resource consumption concerns. The primary resource concern with packet recirculation is bandwidth usage. Excessive packet aggregation can exceed the bandwidth capacity of a programmable switch's loopback interface. Each truncated packet consists of a 14-byte header and 64 bytes of DDoS keywords (Figure 7). After the ith recirculation, the packet size grows to $14+64\cdot i$ bytes. Additionally, during N recirculations, the original truncated packets traverse the loopback interface, consuming $(14+64)\cdot N$ bytes of bandwidth. Therefore, the cumulative bandwidth consumption for N packets with N circulation is calculated as follows:

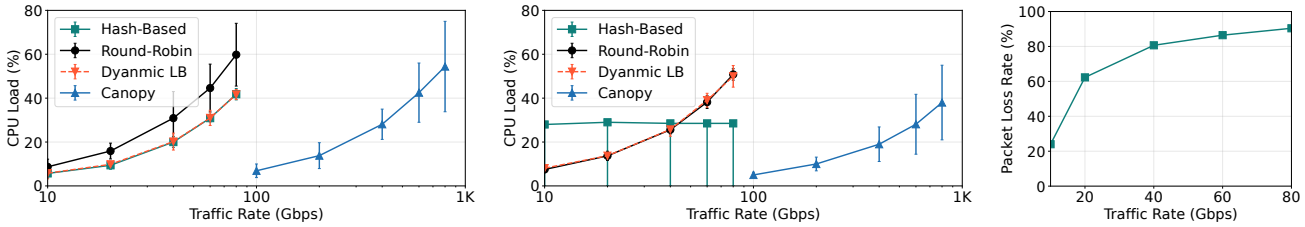
$$\sum_{i=1}^{N}(14+64i+14+64) = (92N+64\cdot\frac{N(N-1)}{2})\times 8 \text{ bps}. \quad (2)$$

For example, with 8 recirculations, compressing 8 packets into one consumes approximately 20 Kb of bandwidth. If the packet arrives at the rate of 1 Gpps and the sampling rate is 10:1, then the resulted bandwidth at the programmable switch's loopback interface will be $1\,\text{Gpps}/10/8\cdot 20\,\text{Kb} = 250\,\text{Gbps}$. But the detection server will receive compressed packets at the rate of $1\,\text{Gpps}/10/8 = 12.5\,\text{Mpps}$.

Therefore, a larger N is not always better. While a larger N allows more packets to be assembled into a single packet, which is beneficial for downstream detection servers, it also increases the number of loopback iterations. This causes the loopback packets to grow larger, consuming more bandwidth on the loopback port. If N exceeds a certain threshold, it may lead to packet loss at the loopback port.

4 Implementation

Dynamic load balancing at programmable switch. We determine the allocation of detection servers and the specific processes within those servers using the destination IP and destination port fields in UDP packets. These values are dynamically computed by our dynamic hash module (§ 3.2), with load-balancing decisions updated every 100 ms, matching the counter refresh interval. The update interval exhibits a trade-off: longer intervals delay the capture of elephant flows, while shorter intervals increase sensitivity to transient traffic bursts and the risk of misclassification. Guided by operational experience, we set the update frequency to 100 ms and find the system largely insensitive to small deviations from this value. Similarly, the elephant-flow threshold T is chosen based on



(a) Detection servers' CPU load with normal flows. *Dynamic LB* refers to Canopy without traffic compression.

(b) Detection servers' CPU load with an elephant flow.

(c) Packet loss; elephant flows; hash-based load balancer.

Figure 10: Evaluation of load balancing schemes and packet processing optimization.

operational practice and also exhibits low sensitivity, so we omit further benchmarks for brevity.

The workflow begins with optical traffic splitting, directing traffic into the programmable switch's data plane. Canopy then performs sampling, DDoS keyword extraction, and packet assembly (see § 3.3). The aggregated and compressed feature packets are forwarded via the switch's internal interconnect module to the control-plane load-balancing module, which module computes the appropriate destination server for each packet based on Equation 1. Once the load-balancing decision is made, the packets are routed back to the data plane via the internal interconnect and transmitted to the designated downstream detection servers.

Detection server optimization. When (compressed) packets arrive at the detection servers, dedicated CPU threads handle packet reception using DPDK, efficiently buffering them into shared I/O queues. These queues act as staging areas for subsequent deep analysis and detection tasks, such as flow correlation, statistics aggregation, and anomaly identification.

To optimize performance, we exploit Non-Uniform Memory Access (NUMA), a memory architecture for modern multi-CPU, multi-core systems. NUMA partitions the system into nodes, each with its own CPU cores and local memory. Local memory access is fast, whereas accessing memory from a remote node incurs higher latency and bandwidth overhead.

By default, these hardware details are abstracted from developers, which can cause unintended cross-node memory access in multiprocess systems, especially during interprocess communication. To address this, Canopy adopts a NUMA-aware design. Upon packet arrival, the NIC uses Direct Memory Access (DMA) to place traffic in the memory of the NUMA node tied to the local CPU. We further structure the code so each CPU core accesses only its local memory. This design avoids costly cross-node access, even in inter-process communication, thereby improving efficiency.

Failure recovery. Canopy employs multiple programmable switches to sustain traffic volumes up to 100 Tbps. Processing capacity is provisioned slightly above peak demand, ensuring redundancy. Upon a switch failure, traffic is seamlessly redistributed across the remaining switches, with packets still forwarded to the same detection servers since all switches share the same configuration. When a detection server fails, the switches promptly update their forwarding tables to redirect traffic to available servers. The entire process—from failure detection to switch update—completes within a subsecond.

Because DDoS detection operates on continuous traffic sampling, with traffic characteristics refreshed every second and attacks generally persisting for longer periods, the temporary loss of a small number of packets during failure detection and table reconfiguration has a negligible impact on overall detection accuracy. As a result, Canopy achieves robustness and high availability in large-scale operational environments.

5 Evaluation

This section first evaluates the two core components of Canopy on a testbed: its dynamic load-balancing mechanism (§ 5.1), and DDoS key extraction and traffic compression techniques (§ 5.2). We then present operational insights from deploying Canopy in production environments (§ 5.3).

5.1 Dynamic Load Balance

Next, we focus on the design of dynamic load balancing. We conducted a local experiment using a 48-core (192-thread) AMD EPYC server as the detection server, paired with a Tofino programmable switch as the load balancer. On the server, 32 threads were allocated for DPDK packet reception, while the remaining threads handled analysis tasks. Since load balancing operates on a per-thread basis, this setup effectively represents 32 independent receivers. We evaluated two types of traffic: (i) normal flows, which follow the distribution observed in our production network, and (ii) elephant flows. The packet length was set to 300B, matching the average observed in our production network.

Figure 10(a) shows the CPU load[5] on the detection servers under normal flow conditions excluding any elephant flows. Dynamic load balancing performs nearly as well as hash-based load balancing and significantly outperforms round-robin load balancing. The round-robin approach increases

[5]The figure depicts the load on CPU threads dedicated to analysis. Note that the CPU threads for DPDK packet reception operate at 100% load at all times, as they function in a busy-waiting manner. But we can tell whether they are overloaded by inspecting if there are any packet drops.

CPU load because it disperses packets with different IP addresses across multiple receivers, requiring additional effort from analysis threads to regroup them.

When handling elephant flows (Figure 10(b)), Canopy's dynamic load balancing achieves performance comparable to round-robin load balancing. In contrast, hash-based load balancing exhibits uneven CPU utilization: some cores are underutilized (around 30% load), while others remain idle. This imbalance leads to inefficiency, as the active CPU threads cannot handle the entire workload. Figure 10(c) highlights the corresponding packet loss rates, which escalate with increasing traffic rates – from approximately 20% at 10 Gbps to nearly 90% at 80 Gbps. Notably, no packet loss occurs in other scenarios, such as normal flows or elephant flows managed by alternative load balancing schemes.

In summary, dynamic load balancing combines the strengths of hash-based and round-robin approaches, adapting effectively to varying traffic demands and achieving optimal performance across diverse conditions.

5.2 Traffic Compression

Next, we evaluate the impact of our DDoS keyword extraction and compression design. We use a setup similar to the load balancing experiment, with dynamic load balancing applied throughout.

First, we examine how packet compression affects packet reception at the detection server. Figure 11(a) shows the packet reception rate for different numbers of packet circulations – *i.e.,* the number of packets from which DDoS keywords are extracted and compressed into a single packet by the programmable switch. A value of 0 indicates that original packets are sent directly to the detection server. The results show that compression improves packet reception: the server can process approximately 36M uncompressed packets, but with compression, this increases to 41M packets at a compression rate of 16. The server's processing capacity improves slightly as the compression rate increases.

Since each compressed packet carries more information than an uncompressed one, it represents a significantly larger volume of original bandwidth, *i.e.,* the bandwidth of actual traffic entering the data center. Figure 11(b) illustrates that without compression, a detection server can handle less than 100 Gbps, whereas with compression, it can process over 1 Tbps of traffic. As a result, traffic compression enables Canopy to handle an order of magnitude larger traffic volume, as shown in Figure 10.

However, these benefits come at a cost. One such cost is added latency. Figure 11(c) shows the additional latency introduced by the programmable switch as a function of the number of packet circulations. The latency increases linearly, with each circulation adding approximately 200ns. Nevertheless, this added latency is negligible compared to the total latency, which is on the order of seconds, with analysis being the primary bottleneck (see Figure 12(b)).

Instead, the most critical cost is bandwidth consumption. Figure 11(d) illustrates the bandwidth demands on the programmable switch's loopback interface as a function of the traffic compression rate N and a sampling rate of 10:1,[6] with the switch capable of processing packets at 1.2 Gpps. The bandwidth demand increases linearly with N, ranging from ∼88Gbps to around 1Tbps. Our switch has 16 loopback ports, 4 of which are dedicated to connecting the dynamic load balancer at the CPU, leaving 12 ports available for packet recirculation. Each port supports 100 Gbps, resulting in a total capacity of 1.2Tbps. Consequently, the maximum feasible compression rate is 36; however, for reliability considerations, we set $N = 16$ in our production network.

Finally, as described in § 4, we implemented a system optimization leveraging NUMA. In our experimental setup, each server is equipped with two NICs, each bound to 16 CPU threads dedicated to DPDK packet reception. We sent uncompressed packets directly to the server via the programmable switch. Figure 12(a) shows the performance difference before and after optimization. The optimization improves performance by 17% to 23%, with greater gains observed as the length of incoming packets increases.

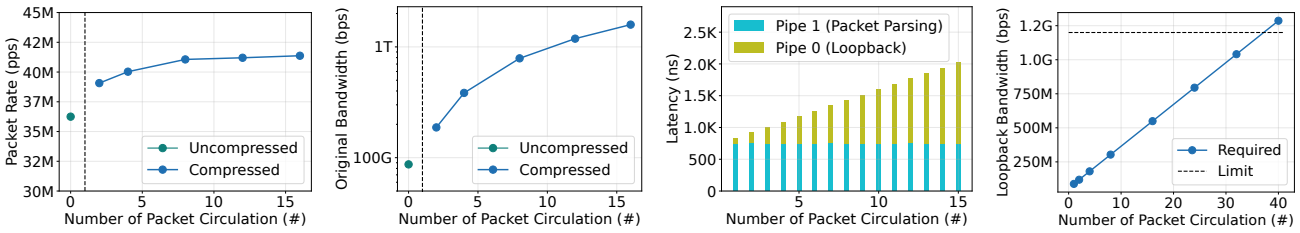
5.3 Operational Experience

Gray-scale deployment. Before fully transitioning from the previous DDoS detection system which relied solely on detection servers without programmable switches, we conducted a gray-scale deployment. In this setup, both Canopy and the legacy DDoS detection system operated simultaneously, each processing all portions of the traffic duplicated from online. This allowed us to compare the two systems and identify any discrepancies and thereby any bugs.

During this testing phase, we encountered a traffic loop issue caused by a software misconfiguration. Certain IP addresses experienced continuous traffic circulation within the DDoS detection systems, resulting in packets being counted multiple times. This led to a dramatic surge in traffic for the affected IPs. According to Canopy's recordings, one IP's traffic spiked to 1.5 Tbps. Canopy identified this anomaly – interpreted as a DDoS attack – within one second and issued a blocking action within the next second, preventing any damage. In contrast, the legacy DDoS detection system failed to handle the traffic spike effectively. Overwhelmed by the load, its detection servers dropped a significant portion of packets, registering only an 80 Gbps traffic increase and failing to trigger a DDoS alarm. This oversight could have allowed a real DDoS attack to bypass detection entirely.

This incident highlights Canopy's capability to effectively manage massive traffic spikes, demonstrating its superiority

[6] Canopy supports arbitrary, runtime-adjustable sampling rates. In practice, a 10:1 rate proved sufficient for most operational needs, with negligible impact on detection accuracy. We omit sensitivity analysis for brevity.



(a) Packet processing rate at detection server.

(b) Original bandwidth at detection server.

(c) Forwarding latency at programmable switch.

(d) Loopback interface bandwidth.

Figure 11: Processing capability of the programmable switch.

over the legacy system.

Overall performance. Canopy has been operational in our production network for over a year, providing a robust and scalable solution for DDoS detection. Currently, the platform supports 25 distinct DDoS detection tasks provided by different users, each leveraging different algorithms and input configurations to adapt to evolving attack patterns.

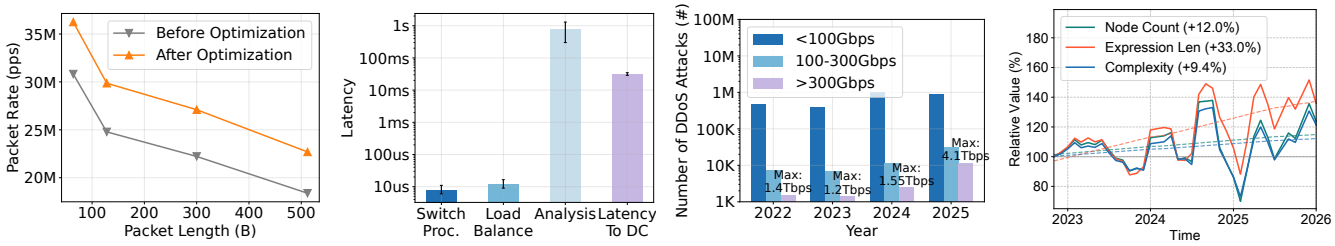
On a daily basis, Canopy handles traffic rate ranging from tens of terabytes to peaks of over 100 Tbps, as illustrated in Figure 3. Despite such extreme conditions, Canopy performs reliably. Figure 12(b) illustrates the DDoS detection latency over the same period, measured using periodic probing traffic. Overall, latency across components remained stable. The switch processing and load-balancing modules consistently achieved sub-10μs latency, benefiting from the line-rate processing of programmable switches. In contrast, the DDoS analysis on the detection server, which includes string matching, grouping, and statistical computations, exhibited the highest latency, ranging from 0.3s to 1.3s. This variance is primarily due to two factors: (i) workload differences and (ii) varying arrival times, as the system processes data in batches, similar to Spark Streaming [2]. Once a DDoS attack is detected, Canopy signals the production network, a process that takes approximately 30 ms. Overall, the detection server accounts for the majority of the processing time, while the programmable switch and notification steps contribute only a negligible fraction in comparison.

Failure modes and mitigation. Despite overall stability, the deployment did encounter hardware-related issues, primarily involving the programmable switch and its internal connectivity to the host CPU. In particular, the P4 switch and CPU are connected via internal 100 Gbps NIC links, and we observed that, on rare occasions, individual NIC ports failed to initialize correctly or temporarily went down after extended operation. Early in deployment, such events could lead to brief packet loss before manual intervention. To address this, we introduced Link Aggregation Control Protocol (LACP)–based port aggregation, allowing traffic to be automatically redirected when a port failure occurs. After adopting port aggregation, these transient failures no longer resulted in observable packet loss. At a finer granularity, while we did not encounter failures within the P4 ingress or egress pipelines in practice, the system design anticipates such low-probability events by internally grouping ports and applying aggregation-style redundancy, ensuring that single-port or single-path failures do not compromise overall operation.

Recent snapshots. Figure 12(c) summarizes the frequency and scale of DDoS attacks observed in our production environment over the past several years. Most attacks operate at rates below 100 Gbps; however, a non-trivial fraction reach several hundred Gbps, and we increasingly observe burst events exceeding 1 Tbps. Overall attack activity has grown substantially over time: the total number of detected attacks increases from approximately 0.41 million in 2023 to 0.95 million in 2025, representing a 132% increase. This growth is dominated by small-scale attacks below 100 Gbps, which rise by 126% (from roughly 0.40 million to 0.91 million). Although individually limited in impact, their high frequency imposes sustained pressure on network infrastructure. More concerningly, large-scale attacks above 300 Gbps grow by 712%, from about 1.4 thousand to 11.6 thousand events, indicating a marked shift toward highly coordinated, bandwidth-intensive attacks. Consistent with this trend, the annual peak attack rate increases from 1.2 Tbps in 2023 to 4.1 Tbps in 2025 (a 242% increase), suggesting that production networks have entered the terabit-scale attack regime.

Motivated by the rapid growth in both the frequency and peak intensity of large-scale attacks, Canopy was deployed in 2025, coinciding with a period when high-bandwidth attacks became significantly more prevalent. Since its deployment, Canopy has had a tangible impact on improving user experience and network security. Before its implementation, when only the detection server was in place, we received an average of fewer than 10 reports or complaints per month regarding undetected DDoS attacks. Since Canopy's deployment, we have received zero such reports, demonstrating its effectiveness in reliably detecting and mitigating attacks. This result underscores the system's success in strengthening network defense mechanisms and enhancing overall user satisfaction.

Evolution of DDoS attacks. Finally, while Canopy focuses on packet processing before the actual DDoS detection algorithms at the detection server, we report our observations on the evolution of DDoS attacks. Recall that an example in § 2 demonstrates that DDoS detection do not rely on a single



(a) NUMA optimization.

(b) Detection delay.

(c) DDoS attacks observed over the past few years (Canopy from 2025).

(d) Evolution of the complexity of DDoS attacks.

Figure 12: End-to-end evaluation of Canopy.

factor. We model each detection algorithm as a processing DAG, where nodes correspond to individual processing or decision operators (e.g., data ingestion, feature extraction, aggregation, filtering, and output), and edges represent data-flow dependencies. We summarize pipeline complexity using four lightweight metrics: the number of operators (NO), the number of directed dependencies (NE), the number of distinct operator types (NT), and the total configuration expression length across all operators, which approximates the semantic density of detection logic. We further define a simple aggregate complexity score $C = \alpha \cdot NO + \beta \cdot NE + \gamma \cdot NT$ to highlight detection algorithm sophistication.

Figure 12(d) plots the multi-year averages of operator count, expression length, and overall complexity for DDoS detection pipelines. Despite year-to-year fluctuations, we observe a clear upward trend: the number of operators increases by 12%, while expression length grows by 33%, leading to a 9.4% increase in overall complexity. This trend indicates that modern detection pipelines increasingly demand CPU resources. By offloading packet parsing, feature extraction, and compression to programmable switches, Canopy reduces the computational burden on detection servers, allowing them to concentrate on flexible and compute-intensive analysis stages.

6 Discussions and Limitations

Feature updates and runtime programmability. The selection of DDoS features (Figure 7) is informed by extensive operational experience. In practice, these features have proven effective in meeting detection requirements and tend to remain stable over time, with updates primarily driven by the emergence of new attack vectors rather than routine tuning. When feature updates are required, the programmable switch program must be recompiled, since feature selection in our design is closely tied to the packet parser (§ 3.3).

Recent work on *runtime-programmable switches* demonstrates that packet-processing logic can be updated dynamically without disrupting live traffic by synthesizing stepwise transition plans that preserve correctness and respect hardware resource constraints [22, 38, 47]. This capability is attractive for maintaining and evolving DDoS-related functionality deployed on programmable switches [50]. However, in contrast to prior approaches that offload detection logic to the switch [50], Canopy deliberately avoids pushing complete or even partial detection algorithms into the data plane. Besides limited support for complex operators, this choice reflects production reliability concerns: faults or misconfigurations in data-plane logic can have an immediate and wide blast radius, potentially affecting service availability. In addition, as discussed in § 5.3, real-world deployments must also contend with hardware-level failures, such as transient port or link failures, which are orthogonal to software update mechanisms. As a result, ensuring robust operation requires backup and failover mechanisms beyond runtime programmability alone. That said, we leave a systematic exploration of such offloading to future work.

SmartNIC versus programmable switch. SmartNICs, including FPGA-based models like Innova and Marvell, as well as ARM-based models like NVIDIA BlueField, present a potential alternative to programmable switches. However, as discussed in § 3.3, load balancing often spans multiple servers, making programmable switches a more natural choice. That said, a hybrid approach – combining programmable switches with SmartNICs – could be a viable solution, with SmartNICs assisting in CPU offloading. Below, we explore the trade-offs between these technologies.

FPGA-based SmartNICs excel at handling complex logic but come with significant limitations. First, their bandwidth is capped at around 200 Gbps per card, far below the 3Tbps+ offered by P4-based programmable switches. Scaling to higher bandwidths would require deploying multiple cards, which is cost-inefficient. Additionally, updating FPGA logic is cumbersome, involving time-intensive reprogramming, making it difficult to meet the flexibility requirements of our platform. Finally, FPGA power consumption tends to be less predictable, adding another layer of operational complexity.

ARM-based SmartNICs, like BlueField, provide better programmability than FPGA-based models but lack line-rate processing capabilities, making them unsuitable for elephant flows, much like CPUs. Nevertheless, they offer a more power-efficient alternative to general-purpose CPUs for certain tasks [46]. We leave further exploration for future work.

7 Related Work

DDoS Attacks and Detection. Distributed Denial-of-Service (DDoS) attacks have been a persistent and evolving threat in network security for decades and remain one of the most prevalent forms of network intrusion [3]. Attackers have developed a diverse range of methods to execute DDoS attacks, often exploiting different protocols such as ICMP, TCP [29, 40], UDP [35, 44], HTTP [26, 41], and others [25].

Over the years, numerous approaches have been developed to detect DDoS attacks. Among them, a promising trend is to rely on statistical analysis [7, 8, 20, 21, 34, 52] to identify deviations from normal traffic behavior and detect potential attack patterns. Many packet features and their statistical derivations have been utilized. Modern advancements in machine learning have also added another layer of sophistication to these detection mechanisms, enabling systems to better distinguish malicious traffic from legitimate activity [12, 15, 24, 49]. The purpose of Canopy is to provide a platform for cloud users to implement DDoS detection per their demands. Therefore, we place the analytics at commodity servers, which allows arbitrary computation on selected packet header fields, and thus enables all of the existing DDoS detection mechanisms.

It is noteworthy that application-layer DoS attacks, which exploit vulnerabilities in application logic rather than overwhelming network resources, have recently attracted significant attention [1, 17, 31–33, 43, 53]. Detecting such attacks often requires a deep understanding of application behavior and/or operations on encrypted traffic, posing unique challenges. As this work focuses on network-layer DDoS detection in cloud environments, application-layer attacks are out of scope and we leave them to future work.

Programmable devices for security. With the slowing of Moore's Law, a variety of specialized, programmable devices have been introduced to accelerate network operations and analytics tasks. For example, GPUs have demonstrated significant potential in accelerating pattern matching [27], while FPGAs are widely used to speed up computational tasks involved in DDoS detection [9, 14, 16, 18, 37, 42]. More recently, programmable switches have gained popularity due to their line-rate processing capabilities, enabling them to handle various network tasks, including DDoS detection [7, 8, 28, 34, 50]. For example, Poseidon [50] proposed a set of defense primitives for programmable switches, while Peregrine explored implementing a new feature extraction [8].

Unlike prior work [8, 50], which uses programmable switches for computation, our approach leverages them primarily as packet parsers and traffic compressors. This design emphasizes flexibility in detection mechanisms—a critical requirement for adapting to the dynamic demands of production environments. Moreover, prior systems often assume [50, 52] that detection directly intercepts production traffic, allowing programmable devices to drop malicious packets inline. In practice, however, production networks typically employ passive traffic duplication (see Figure 1) to avoid disrupting live traffic; when malicious traffic is detected, the system alerts the data center rather than acting directly.

Load balance. Load balancing is a fundamental component of computer networks, widely utilized across various applications and systems [13, 30]. Modern data center network topologies are specifically designed to optimize load balancing [4, 23], and it plays a crucial role in enhancing application performance [19, 39, 45]. Innovations in load balancing, such as incorporating deterministic approaches [51], have further advanced its capabilities. However, many of these methods require specialized setups, such as feedback from servers [36] or understanding underlying network [51], making them unsuitable for our use case. Another trend of study is dynamic load balancing, which has been extensively studied [5, 13].

Concerning packet processing, RSS++ [10] and state-compute replication [48] explore improving performance by managing or replicating state under relatively strict execution models – either within a single machine or by enforcing global ordering for individual stateful flows. In contrast, our work targets distributed DDoS detection pipelines, where state is inherently multi-dimensional, spans many flows, and must be aggregated across servers without assuming tight ordering.

In this paper, we present the first detailed exploration of dynamic load balancing for DDoS detection in production environments. We show that basic load-balancing schemes alone are insufficient to meet the demands of real-world DDoS detection, and instead propose a hybrid approach. Our system is implemented in a production-grade deployment and has been successfully operated at traffic scales of hundreds of terabits per second.

8 Conclusion

This paper presents our experience with Canopy, a scalable DDoS detection system capable of handling traffic of over 100 Tbps. Canopy introduces two key innovations: a dynamic load balancer and a traffic compression mechanism on programmable switches, both designed to ease server workloads. Our evaluation shows that the load balancer effectively manages long-tailed traffic, while traffic compression greatly increases server processing capacity. We also share insights into traffic patterns and operational performance from real-world deployment. Overall, Canopy meets our design goals and has been running seamlessly in production.

9 Acknowledgments

We sincerely thank our shepherd Srinivas Narayana and anonymous reviewers for their insightful comments. We also thank software engineers at Tencent for their support in deploying Canopy in production. Congcong Miao is the corresponding author.

References

[1] Application Layer DDoS Attack. https://www.cloudflare.com/learning/ddos/application-layer-ddos-attack/. Last accessed in January 2025.

[2] Spark Structured Streaming | Apache Spark. https://spark.apache.org/streaming/. Last accessed in January 2025.

[3] The cyber surge: Kaspersky detected 467,000 malicious files daily in 2024. https://www.kaspersky.com/about/press-releases/the-cyber-surge-kaspersky-detected-467000-malicious-files-daily-in-2024. Last accessed in January 2025.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of ACM SIGCOMM, Seattle, WA, USA, August 17-22, 2008*, pages 63–74. ACM, 2008.

[5] A. M. Alakeel et al. A guide to dynamic load balancing in distributed computer systems. *International journal of computer science and information security*, 10(6):153–160, 2010.

[6] M. Alasmar, G. Parisis, R. G. Clegg, and N. Zakhleniuk. On the distribution of traffic volumes in the internet and its implications. In *Proceedings of IEEE INFOCOM, Paris, France, April 29 - May 2, 2019*, pages 955–963. IEEE, 2019.

[7] A. G. Alcoz, M. Strohmeier, V. Lenders, and L. Vanbever. Aggregate-based congestion control for pulse-wave ddos defense. In *Proceedings of ACM SIGCOMM, August 22 - 26, 2022*, pages 693–706. ACM, 2022.

[8] J. R. Amado, F. Pereira, D. Pissarra, S. Signorello, M. Correia, and F. M. V. Ramos. Peregrine: Ml-based malicious traffic detection for terabit networks. *CoRR*, abs/2403.18788, 2024.

[9] Z. K. Baker and V. K. Prasanna. High-throughput linked-pattern matching for intrusion detection systems. In *Proceedings of ANCS, Princeton, New Jersey, USA, October 16-18, 2005*, pages 193–202. ACM, 2005.

[10] T. Barbette, G. P. Katsikas, G. Q. M. Jr., and D. Kostic. RSS++: load and state-aware receive side scaling. In *Proceedings of CoNEXT, Orlando, FL, USA, December 09-12, 2019*, pages 318–333. ACM, 2019.

[11] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of ACM IMC, Melbourne, Australia - November 1-3, 2010*, pages 267–280. ACM, 2010.

[12] A. L. Buczak and E. Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun. Surv. Tutorials*, 18(2):1153–1176, 2016.

[13] V. Cardellini, M. Colajanni, and P. S. Yu. Dynamic load balancing on web-server systems. *IEEE Internet Comput.*, 3(3):28–39, 1999.

[14] M. Ceska, V. Havlena, L. Holík, J. Korenek, O. Lengál, D. Matousek, J. Matousek, J. Semric, and T. Vojnar. Deep packet inspection in fpgas via approximate nondeterministic automata. In *27th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), San Diego, CA, USA, April 28 - May 1, 2019*, pages 109–117. IEEE, 2019.

[15] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki. Network intrusion detection for iot security based on learning techniques. *IEEE Commun. Surv. Tutorials*, 21(3):2671–2701, 2019.

[16] C. R. Clark and D. E. Schimmel. Scalable pattern matching for high speed networks. In *12th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 20-23 April 2004, Napa, CA, USA, Proceedings*, pages 249–257. IEEE Computer Society, 2004.

[17] S. A. Crosby and D. S. Wallach. Denial of service via algorithmic complexity attacks. In *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*. USENIX Association, 2003.

[18] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood. Deep packet inspection using parallel bloom filters. *IEEE Micro*, 24(1):52–61, 2004.

[19] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein. Maglev: A fast and reliable software network load balancer. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Santa Clara, CA, USA, March 16-18, 2016*, pages 523–535. USENIX Association, 2016.

[20] D. Erhan and E. Anarim. Hybrid ddos detection framework using matching pursuit algorithm. *IEEE Access*, 8:118912–118923, 2020.

[21] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred. Statistical approaches to ddos attack detection and response. In *3rd DARPA Information Survivability Conference and Exposition (DISCEX-III 2003), 22-24 April 2003, Washington, DC, USA*, page 303. IEEE Computer Society, 2003.

[22] Y. Feng, Z. Chen, H. Song, W. Xu, J. Li, Z. Zhang, T. Yun, Y. Wan, and B. Liu. Enabling in-situ programmability in network data plane: From architecture to language. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Renton, WA, USA, April 4-6, 2022*, pages 635–649. USENIX Association, 2022.

[23] A. G. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *Proceedings of ACM SIGCOMM, Barcelona, Spain, August 16-21, 2009*, pages 51–62. ACM, 2009.

[24] A. Hamza, H. H. Gharakheili, T. A. Benson, and V. Sivaraman. Detecting volumetric attacks on lot devices via sdn-based monitoring of MUD activity. In *Proceedings of the 2019 ACM Symposium on SDN Research (SOSR), San Jose, CA, USA, April 3-4, 2019*, pages 36–48. ACM, 2019.

[25] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita. Botnet in ddos attacks: Trends and challenges. *IEEE Commun. Surv. Tutorials*, 17(4):2242–2270, 2015.

[26] G. A. Jaafar, S. M. Abdullah, and S. A. Ismail. Review of recent detection methods for HTTP ddos attack. *J. Comput. Networks Commun.*, 2019:1283472:1–1283472:10, 2019.

[27] M. A. Jamshed, J. Lee, S. Moon, I. Yun, D. Kim, S. Lee, Y. Yi, and K. Park. Kargus: a highly-scalable software-based intrusion detection system. In *the ACM Conference on Computer and Communications Security (CCS), Raleigh, NC, USA, October 16-18, 2012*, pages 317–328. ACM, 2012.

[28] T. Jepsen, D. Álvarez, N. Foster, C. Kim, J. Lee, M. Moshref, and R. Soulé. Fast string searching on PISA. In *Proceedings of the 2019 ACM Symposium on SDN Research (SOSR), San Jose, CA, USA, April 3-4, 2019*, pages 21–28. ACM, 2019.

[29] M. Kührer, T. Hupperich, C. Rossow, and T. Holz. Hell of a handshake: Abusing TCP for reflective amplification ddos attacks. In *8th USENIX Workshop on Offensive Technologies (WOOT), San Diego, CA, USA, August 19, 2014*. USENIX Association, 2014.

[30] P. Kumar and R. Kumar. Issues and challenges of load balancing techniques in cloud computing: A survey. *ACM Comput. Surv.*, 51(6):120:1–120:35, 2019.

[31] Y. Li, Z. Chen, J. Cao, Z. Xu, Q. Peng, H. Chen, L. Chen, and S. Cheung. Redoshunter: A combined static and dynamic approach for regular expression dos detection. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 3847–3864. USENIX Association, 2021.

[32] Z. Li, H. Jin, D. Zou, and B. Yuan. Exploring new opportunities to defeat low-rate ddos attack in container-based cloud environment. *IEEE Trans. Parallel Distributed Syst.*, 31(3):695–706, 2020.

[33] S. Lin, S. Chen, Y. Xiao, Y. Gu, A. Kuzmanovic, and X. Yang. Preacher: Secure and practical password preauthentication by content delivery networks. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI), Philadelphia, PA, USA, April 28-30, 2025*, pages 1399–1419. USENIX Association, 2025.

[34] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar. Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches. In *30th USENIX Security Symposium, August 11-13, 2021*, pages 3829–3846. USENIX Association, 2021.

[35] J. Mirkovic, G. Prier, and P. L. Reiher. Attacking ddos at the source. In *10th IEEE International Conference on Network Protocols (ICNP), 12-15 November 2002, Paris, France, Proceedings*, pages 312–321. IEEE Computer Society, 2002.

[36] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distributed Syst.*, 12(10):1094–1104, 2001.

[37] P. Orosz, T. Tothfalusi, and P. Varga. Fpga-assisted DPI systems: 100 gbit/s and beyond. *IEEE Commun. Surv. Tutorials*, 21(2):2015–2040, 2019.

[38] Y. Qiu, R. Beckett, and A. Chen. Synthesizing runtime programmable switch updates. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Boston, MA, April 17-19, 2023*, pages 613–628. USENIX Association, 2023.

[39] M. A. Qureshi, Y. Cheng, Q. Yin, Q. Fu, G. Kumar, M. Moshref, J. Yan, V. Jacobson, D. Wetherall, and A. Kabbani. PLB: congestion signals are simple and effective for network load balancing. In *Proceedings of ACM SIGCOMM, Amsterdam, The Netherlands, August 22 - 26, 2022*, pages 207–218. ACM, 2022.

[40] C. L. Schuba, I. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni. Analysis of a denial of service attack on TCP. In *1997 IEEE Symposium on Security and Privacy, May 4-7, 1997, Oakland, CA, USA*, pages 208–223. IEEE Computer Society, 1997.

[41] K. Singh, P. Singh, and K. Kumar. Application layer HTTP-GET flood ddos attacks: Research landscape and challenges. *Comput. Secur.*, 65:344–372, 2017.

[42] H. Song, T. S. Sproull, M. Attig, and J. W. Lockwood. Snort offloader: A reconfigurable hardware NIDS filter. In *Proceedings of the 2005 International Conference on Field Programmable Logic and Applications (FPL), Tampere, Finland, August 24-26, 2005*, pages 493–498. IEEE, 2005.

[43] C. Staicu and M. Pradel. Freezing the web: A study of redos vulnerabilities in javascript-based web servers. In *27th USENIX Security Symposium, Baltimore, MD, USA, August 15-17, 2018*, pages 361–376. USENIX Association, 2018.

[44] D. R. Thomas, R. Clayton, and A. R. Beresford. 1000 days of UDP amplification ddos attacks. In *2017 APWG Symposium on Electronic Crime Research, eCrime 2017, Phoenix, AZ, USA, April 25-27, 2017*, pages 79–84. IEEE, 2017.

[45] D. Wetherall, A. Kabbani, V. Jacobson, J. Winget, Y. Cheng, C. B. M. III, U. P. Moravapalle, P. Gill, S. Knight, and A. Vahdat. Improving network availability with protective reroute. In *Proceedings of ACM SIGCOMM, New York, NY, USA, 10-14 September 2023*, pages 684–695. ACM, 2023.

[46] Y. Xiao, D. Z. Tootaghaj, A. Dhakal, L. Cao, P. Sharma, and A. Kuzmanovic. Conspirator: Smartnic-aided control plane for distributed ML workloads. In *Proceedings of the 2024 USENIX Annual Technical Conference (ATC), Santa Clara, CA, USA, July 10-12, 2024*, pages 767–784. USENIX Association, 2024.

[47] J. Xing, K. Hsu, M. Kadosh, A. Lo, Y. Piasetzky, A. Krishnamurthy, and A. Chen. Runtime programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Renton, WA, USA, April 4-6, 2022*, pages 651–665. USENIX Association, 2022.

[48] Q. Xu, S. Miano, X. Gao, T. Wang, A. Murugadass, S. Zhang, A. Sivaraman, G. Antichi, and S. Narayana. State-compute replication: Parallelizing high-speed stateful packet processing. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI), Philadelphia, PA, USA, April 28-30, 2025*, pages 937–957. USENIX Association, 2025.

[49] K. Yang, S. Kpotufe, and N. Feamster. An efficient one-class SVM for anomaly detection in the internet of things. *CoRR*, abs/2104.11146, 2021.

[50] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu. Poseidon: Mitigating volumetric ddos attacks with programmable switches. In *27th Annual Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[51] Z. Zhang, H. Zheng, J. Hu, X. Yu, C. Qi, X. Shi, and G. Wang. Hashing linearity enables relative path control in data centers. In *Proceedings of the 2021 USENIX Annual Technical Conference (ATC), July 14-16, 2021*, pages 855–862. USENIX Association, 2021.

[52] Z. Zhao, H. Sadok, N. Atre, J. C. Hoe, V. Sekar, and J. Sherry. Achieving 100gbps intrusion prevention on a single server. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Virtual Event, November 4-6, 2020*, pages 1083–1100. USENIX Association, 2020.

[53] M. Zolotukhin, T. Hämäläinen, T. Kokkonen, and J. Siltanen. Increasing web service availability by detecting application-layer ddos attacks in encrypted traffic. In *23rd International Conference on Telecommunications, ICT 2016, Thessaloniki, Greece, May 16-18, 2016*, pages 1–6. IEEE, 2016.