A Survey on Security Applications of P4 Programmable Switches and a STRIDE-based Vulnerability Assessment

Ali AlSabeh[a], Joseph Khoury[b], Elie Kfoury[a], Jorge Crichigno[a], Elias Bou-Harb[b]

[a] *College of Engineering and Computing, University of South Carolina, Columbia, USA*
[b] *The Cyber Center For Security and Analytics, University of Texas at San Antonio, USA*

Abstract

The emergence of the IoT, cloud systems, data centers, and 5G networks is increasing the demand for a rapid development of new applications and protocols at all levels of the protocol stack. However, traditional fixed-function data planes have been characterized by a lengthy and costly development process at the hand of few chip manufacturers. Recently, data plane programmability has attracted significant attention, permitting network owners to run customized packet processing functions using P4, the *de facto* data plane programming language. Network security is one of the key research areas exploiting the capabilities of programmable switches. Examples include new encapsulations and secure tunnels implemented in short times, mitigation techniques for DDoS attacks that occur at terabit rates, customized firewalls that track hundreds of thousands of connections per second, and traffic anonymization systems that operate at line rate. Moreover, applications can be reconfigured in the field without additional hardware upgrades, facilitating the deployment of new defenses against unforeseen attacks and vulnerabilities. Furthermore, these security applications are designed by network owners who can meet their specific requirements, rather than by chip manufacturers.

Despite the impressive advantages of programmable data plane switches, the literature has been missing a comprehensive survey on security applications. To this end, this paper provides a concise background on programmable switches and their main features that are relevant to security. It then presents a taxonomy that surveys, classifies, and analyzes articles related to security applications developed with P4. Additionally, the paper employs a STRIDE analysis to examine vulnerabilities related to general P4 applications (e.g., congestion control, load balancing, in-network cache) and proposes plausible remediation approaches. Furthermore, challenges associated with programmable data planes, the impact of these challenges on security implementations, and schemes to eliminate or mitigate them are discussed. Finally, the paper discusses future endeavors and open research problems.

Keywords: P4 language, programmable data plane, P4 security applications and implications, STRIDE model, challenges and solutions in P4.

1. Introduction

Computer networks have been vulnerable to a myriad of attacks partially attributed to the lack of flexibility of networking devices (e.g., routers, switches, security appliances). The traditional closed-design paradigm has limited the data plane capability to proprietary implementations which are hard-coded by chip manufacturers, thus preventing reprogrammability and hindering new defenses against zero-day attacks. Such limitations have bolstered the ascendance of the Software-Defined Networking (SDN) [1]. While SDN reduced network complexity and spurred control plane innovation at the speed of software development, it is restricted to the OpenFlow specifications and the fixed-function data plane [2]. Consequently, attack mitigation is mainly offloaded to the control plane or to a third-party middlebox that operates at significantly lower speeds than the data plane. This has pushed the research community to continuously extend the OpenFlow data plane specification, as well as devise new data plane abstractions [3, 4]. For instance, OpenState [5] add stateful data plane programming to OpenFlow using eXtended Finite State Machines (XFSM) to do include some intelligence in the switch.

Large scale attacks such as Denial of Service (DoS) and Distributed DoS (DDoS) have been ravaging the Internet with peak volume reaching Terabits per second (Tbps). On one hand, software-based defenses are lagging behind, incapable of keeping up with such attack rates. On the other, traffic scrubbing centers, one of the most widely used defenses, deploy expensive and proprietary hardware appliances. Thus, despite their

E-mail addresses: aalsabeh@email.sc.edu (Ali AlSabeh), joseph.khoury@utsa.edu (Joseph Khoury), ekfoury@email.sc.edu (Elie Kfoury), jcrichigno@cec.sc.edu (Jorge Crichigno), elias.bouharb@utsa.edu (Elias Bou-Harb)

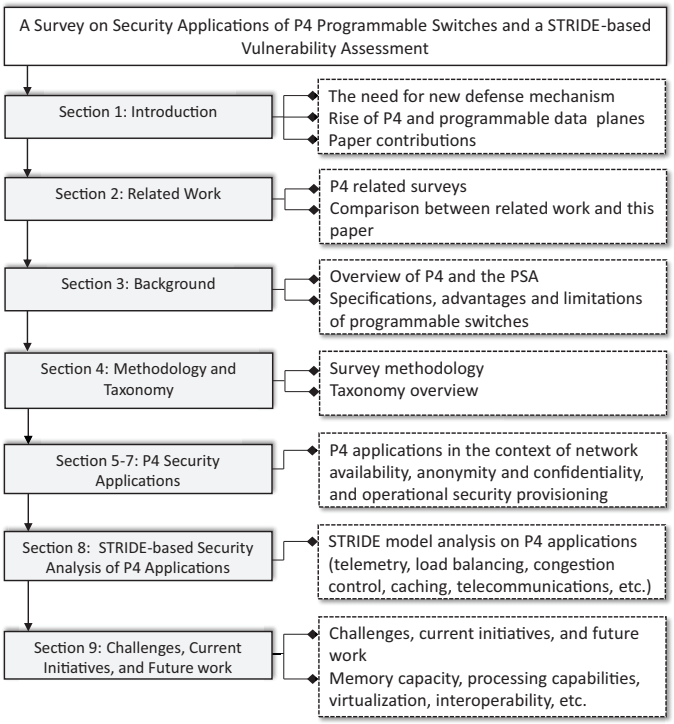

Figure 1: Paper road map.

high performance, they are inflexible in terms of functionality, capacity, and placement location [6].

Additionally, security appliances are limited to operate according to standards [2] and innovation is discouraged by lengthy standardization processes. As an example, after being initially conceived by Cisco and VMware [7], the Application Specific Integrated Circuit (ASIC) implementation of the Virtual Extensible Local Area Network (VXLAN) [8], a simple tunneling protocol, took several years, a process that could have been reduced to weeks by software implementations.

The popularity of cloud systems has fostered the use of virtual security equipment as well [9]. However, such solutions run on general-purpose Central Processing Units (CPUs) that are unable of processing at Tbps rates [10]. Other security and privacy functions face similar challenges. For example, legacy anonymity tools used to preserve privacy (e.g., onion routing) require a series of overlay proxies to redirect traffic with no performance guarantees, thus imposing bandwidth and latency issues [11, 12].

Programmable data plane switches have recently emerged, where the software that describes the behavior of how packets are processed can be conceived, tested, and deployed in a much shorter time span by operators, engineers, researchers, and practitioners in general [13]. The *de facto* language for defining the forwarding behavior is P4 [1], which stands for Programming Protocol-independent Packet Processors. Essentially, P4 programmable switches have removed the entry barrier to network design, previously reserved to chip manufacturers. Enabling users to program the switch ASIC has resulted in unforeseen applications: replacing hundreds of load balancer servers with one programmable switch [14], developing Fifth-Generation (5G) firewalls with General Packet Radio Service (GPRS) tunneling capability [15], devising in-network memory cache that can process over 2 billion queries per second [16], and others.

1.1. Contribution

Despite the wide integration of various security applications (e.g., firewall, DDoS mitigation, traffic anonymization, etc.), even the most recent work does not thoroughly explore security use cases in P4 as shown in Table 1. This survey covers the gap by digging deep into each work and presenting paramount information for security researchers and practitioners. For a clear separation and encompassing of the crucial security objectives, this survey classifies P4 security applications into network availability, anonymity and confidentiality, and operational security provisioning.

Furthermore, with the increased flexibility and programmability, a wide range of security implications that can wreak havoc in the data plane are induced. Nonetheless, no research work encompasses the notable P4 applications (e.g., congestion control, network telemetry, load balancing) and highlights the main attacks on them. Accordingly, this survey explores such applications, infers potential vulnerabilities, and proposes potential mitigation strategies. The contributions of this paper can be summarized as follows:

- An overview of the P4 language and the Portable Switch Architecture (PSA) is presented, along with the advantages and limitations of programmable switches in the context of security.
- A taxonomy of P4 security applications is provided to help enlighten the cybersecurity research directions.
- A Spoofing, Tampering, Repudiation, Information disclosure, DoS, and Elevation of privilege (referred to as the STRIDE model) analysis on main P4 applications is presented. Additionally, plausible remediation solutions are described.
- Challenges related to programmable switches and the corresponding impact of security applications are presented. Additionally, open research problems and future directions are discussed.

1.2. Paper Organization

The road map of this survey is depicted in Fig. 1. Section 2 compares existing surveys on P4 and demonstrates the added value of this work. Section 3 presents an overview of P4 programmable switches and their features, advantages, and limitations. It also contrasts programmable switches with general-purpose CPUs. Section 4 discusses the survey methodology and describes the proposed taxonomy. Sections 5-7 collate and analyze P4 applications that address network availability, anonymity and confidentiality, and operational security provisioning. Section 8 studies main P4 applications not (directly)

Table 1: Comparison with Related Surveys

Paper	P4 Data Plane		P4 Security Applications			Security Implications		Discussion	
	Pipeline	Specs., Pros., Cons.	Background, Literature	Intra-category Comparison	Comparison with Legacy	Vulnerability	Mitigation	Challenges /Initiatives	Future Work
[17]	●	⊙	⊙	⊙	⊙	⊙	⊙	⊙	●
[18]	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	●
[19]	●	⊙	⊙	⊙	⊙	⊙	⊙	●	●
[20]	⊙	⊙	⊙	⊙	⊙	⊙	⊙	●	●
[21]	●	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙
[22]	⊙	⊙	⊙	⊙	⊙	●	⊙	⊙	⊙
[23]	⊙	⊙	⊙	⊙	⊙	●	⊙	⊙	⊙
[24]	○	○	⊙	⊙	○	⊙	⊙	●	●
[25]	●	●	⊙	⊙	○	○	○	●	●
[26]	●	●	⊙	⊙	⊙	⊙	⊙	⊙	⊙
[27]	●	●	⊙	⊙	⊙	⊙	⊙	⊙	●
[28]	●	●	⊙	⊙	⊙	⊙	⊙	●	●
This paper	●	●	●	●	●	●	●	●	●

● Covered ○ Not covered ⊙ Partially covered

related to security (e.g., congestion control, load balancing) and describes security implications using the STRIDE model [29]. Section 9 lists challenges associated with programming switches and their impact on security. It then discusses current initiatives that overcome the challenges and provides a reflection on open research issues, and Section 10 concludes the paper.

2. Related Work

Satapathy [17] presents a report about SDN/OpenFlow and the current challenges in this paradigm. Then, the author introduces the P4 language and provides tutorial-like guidelines to work with P4. Furthermore, a number of P4 use cases and applications are present, such as network telemetry, load balancing, and DDoS detection. The work discusses only three use cases of P4 applications including DDoS detection for security.

Kaljic et al. [18] present a survey that covers topics addressing the aspects of SDN data plane flexibility and programmability. In particular, the survey presents an overview of hardware-based and software-based technologies proposed in standard SDN data plane architectures (e.g., Field-Programmable Gate Array (FPGA)). Furthermore, the authors study the implications of SDN-related research on data plane evolution and generalize four approaches to address the problem of programmability and flexibility in the data plane, namely, programmability (e.g., P4), stateful data plane, deeply programmable networks (e.g., caching, conversion of encoding, Deep Packet Inspection (DPI), etc.), and new data plane architectures. The survey does not discuss security applications and implications in P4.

Cordeiro et al. [19] survey the literature on data plane programmability while focusing on P4 and diving into open research questions from recent related work. Furthermore, the authors survey the opportunities that were harnessed in the programmable data plane, in addition to a number of challenges that accompany it. The survey succinctly discusses intrusion detection and prevention in the data plane and does not present a comprehensive literature on the related work.

Bifulco et al. [20] publish a small survey on programmable data plane abstractions, architectures, and open issues. First, the authors define that a programmable switch must be able to reconfigure the packet processing logic in a systematic, rapid, and comprehensive manner. Second, they present a layered view of the programmable switch architectures. Furthermore, the authors present a brief overview of software and hardware switches, as well as the differences between stateless and stateful data plane architectures. The authors conclude their work with a number of open issues coupled with programmable data planes, such as security and scalability as pinpointed in [30]. The survey lacks a discussion on the security applications and implications in P4.

Stubbe [21] conducts a survey on the different compilers and interpreters used for the P4 language. The author describes the packet processing pipeline in P4 and annotates the major difference between $P4_{14}$ and $P4_{16}$. The survey lists open source and proprietary P4 compilers and interpreters, however, it does not discuss security-related topics in P4.

Agape et al. [22] provide the first STRIDE model analysis to P4. First, the authors specify the assets of the P4 platform (e.g., control plane, table entries, compiler, etc.). Then, a high-level analysis of potential vulnerabilities on the assets is carried based on the STRIDE model. Also, the authors discuss a number of vulnerabilities related to the P4 language and compiler such as inducing an undefined behavior through invalid headers. The work does not discuss potential vulnerabilities in major enabling P4 applications, but rather discusses attack points within some P4 components.

Dumitru et al. [23] observe the actual behavior of P4 targets, namely Behavioral Model (BMv2) software switch [31], P4-NetFPGA [32], and Intel Tofino (formerly Barefoot) switch [33], when they encounter various bugs. Such bugs in P4 programs include accessing/reading/writing invalid headers, loops,

dead packet resurrection, and implicit forwarding behavior. Moreover, the authors find a range of potentially exploitable behaviors that are target-dependent, and succinctly analyze a group of plausible P4 data plane threats based on the STRIDE model [29]. The work does not extensively study application-specific attacks, but rather discusses vulnerabilities in generic P4 programs.

Black et al. [24] discuss adversarial data plane verification techniques that detect anomalies while taking into consideration that the data plane could be compromised. The verification techniques are categorized into those relying on cryptography, packet probing [34], distributed statistics, and hash collection. Such verification techniques are compared based on their attack coverage (e.g., inter-switch collusion, drops redirect, packet injection, etc.) and their performance overhead. Furthermore, the authors discuss unsolved problems and open new horizons for future work in adversarial data plane verification techniques. The work does not comprehensively survey P4 security applications, but rather focuses on the network verification part.

Michel et al. [25] present a survey on programmable data plane abstractions, architectures, algorithms, and applications. Firstly, the survey discusses the available programmable data plane architectures (e.g., ASIC, FPGA, x86) while comparing them based on metrics such as the flexibility and processing speed. Secondly, the survey discusses abstractions in the context of packet parsing and scheduling, stateful processing, programming languages, and compilers. Thirdly, the survey considers relevant algorithms and data structures for packet processing that enhance the performance of the data plane. The authors of the survey compartmentalize P4 applications into categories such as telemetry, in-network computations, consensus, etc., with no emphasis on security applications.

Hauser et al. [26] conduct a survey on data plane programmability with P4, where they organize the work into two parts. The first part contains a thorough overview of the P4 language, architectures, compilers, targets, and data plane Application Programming Interfaces (APIs). The second part of the survey is comprised of P4-based applied research areas, such as monitoring, traffic management and congestion control, routing and forwarding, advanced networking, network security, and other miscellaneous research domains. In the section related to network security, the authors subdivide 38 papers into six categories; namely firewall, port knocking, DDoS mitigation mechanisms, Intrusion Detection System (IDS) and DPI, and connection security. The survey only lists the applications without a thorough discussion of the implementation details, limitations, or comparison between the work.

Kaur et al. [27] provide a comprehensive overview of the P4 language (architecture, compiler, P4 runtime), in addition to the research efforts done in this domain. The authors dissect the efforts into four categories including network monitoring, DDoS attacks detection, load balancing, and packet aggregation and disaggregation. Additionally, they present challenges related to security, cost, and design, etc. The survey lacks extensive literature about the existing security applications in P4.

A recent work by Kfoury et al. [28] present an exhaustive survey on P4 programmable data plane switches. The work analyzes more than 150 articles related to P4 and compartmentalizes them into seven categories, including In-band Network Telemetry (INT), measurements, performance, middleboxes, Internet of Things (IoT), cybersecurity, and network testing. Furthermore, the authors provide current challenges and future trends in P4. The survey does not provide comprehensive details on the discussed P4 security papers. Additionally, the survey does not analyze the security implications of P4 applications.

Novelty of this work: Table 1 summarizes the related work described above and compares them with this survey. The originality of this work is discussed in the concepts below while comparing with the aforementioned P4 surveys.

- *Programmable data plane properties and the corresponding security implications:* surveys such as [24, 27, 28] discuss the features and architectures of P4 without interconnecting them with the security implications. This survey provides a robust background on the programmable data plane while tightly correlating it with various security concepts (e.g., the effect of the fast packet forwarding speed on detecting anomalies). Additionally, this survey compares programmable switches and general-purpose CPUs and discusses the security advantages and limitations in both schemes (e.g., more expressiveness in CPUs, whereas faster attack detection in P4).
- *Security objectives achieved in P4:* to the best of the authors' knowledge, this survey is the first to focus on P4 applications related to the three network security objectives (network availability, anonymity and confidentiality, and operational security provisioning) and provide a microscopic comparison between the works belonging to the same category (intra-category), as well as a comparison of P4-based defenses against legacy server-based defenses. Such categorization is tailored to the P4 security papers presented in the literature and is tightly correlated with the well-known security goals (i.e., confidentiality, integrity, and availability).
- *Technical implementation details for security practitioners:* although P4 surveys such as [27, 28] discuss papers related to cybersecurity, their discussion is tailored to general readers rather than security researchers. This survey provides more comprehensive details and illustrative figures for the related literature than other P4 surveys such as in [26, 28]. For instance, in DDoS attacks, this survey zooms in to the type of attack mitigated (e.g., volumetric, slow DDoS, SYN, etc.), compares them based on the technique used (e.g., pipeline implementation, data structures, deployment infrastructure, etc.) and the notable results obtained. In addition, this paper pinpoints the limitation of every work and provides the lessons learned for each network security objective.
- *STRIDE-based analysis:* papers such as [22] and [23] provide a succinct STRIDE-based analysis on the P4 language and compiler. On the contrary, this survey tailors

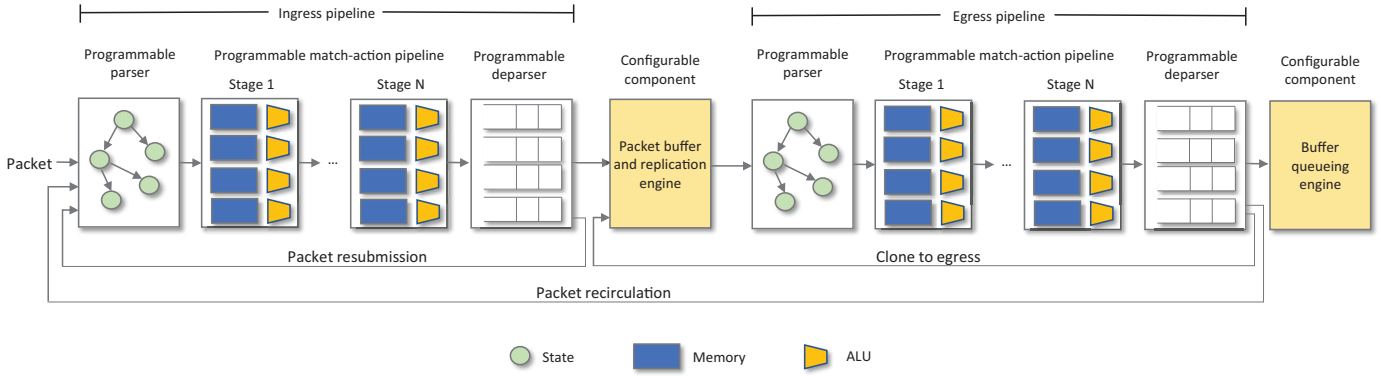

Figure 2: Portable Switch Architecture (PSA) [35]. The parser extracts packet header fields. Pipelines consist of match-action stages, each with memory elements (e.g., SRAM, TCAM) and ALUs. A header field is matched against a value in a table on memory and the matched entry specifies corresponding actions on header field/s. The system operates at line rate (e.g., 100 Gigabits per second (Gbps)) and thus ALUs execute simple operations only (bit manipulation, simple arithmetic). Although complex schemes such as encryption extend beyond the intend of PSA, they have been implemented (see [36]) using recirculation and resubmission to apply further processing over header fields at the expense of lower throughput.

the STRIDE analysis to P4 applications, describes the potential attacks, discusses and proposes possible mitigation approaches, and provides the lessons learned.

- *Challenges and future work pertaining to security:* challenges in P4 are discussed in a number of surveys (e.g., [25, 28]). However, none of the related work provides a thorough discussion on the security implications of such challenges in contrast to this survey. Furthermore, this survey centers the future work on possible security endeavors in P4 (e.g., future work in firewalls, DDoS attacks, cryptography, etc.).

3. Background Information

3.1. P4 Language and the Portable Switch Architecture (PSA)

The match-action paradigm refers to the process by which a switch first matches a header (e.g., Internet Protocol (IP), Transport Control Protocol (TCP)) or non-header (e.g., metadata) field against a value in a table. Then, the matched entry specifies corresponding actions that are applied to the packet header. Consider Fig. 2. The PSA [35] describes the capabilities of switches that can be programmed using $P4_{16}$. The programmable parser enables the programmer to define the headers (according to custom or standard protocols) and parse them. The programmable match-action pipeline executes the operations over the packet headers and intermediate results. A single match-action stage has multiple memory blocks (e.g., Static Random-Access Memory (SRAM) and Ternary Content Addressable Memory (TCAM) tables) and Arithmetic Logic Units (ALUs). The memory blocks are used for matching and the ALUs are the action units. Additional action logic can be utilized through stateful objects (e.g., registers, counters, meters) that are stored in the SRAM. The deparser specifies the packet contents to be sent to the packet buffer and corresponding metadata. The egress pipeline provides additional match-action capability, allows more efficient processing of multicast packets by deferring per-port modifications until after buffering, permits a packet to be recirculated to the ingress pipeline, and enables a packet to be cloned. The components of the ingress and the egress pipelines are the same (memory, ALU, etc.) and thus, in practical implementations, they share the same physical components. Sharing resources reduces implementation cost [3, 37].

PSA encourages portability; as long as the P4 code conforms to PSA, the code must run on many platforms that adhere to PSA [35]. Implementations may include a superset of PSA primitives (e.g., Tofino Native Architecture [33]).

3.2. Domain-specific Processor for Networking

PSA is a model of a domain-specific processor for networking. PSA implementations are designed to execute packet header operations efficiently. Features include:

- *Memory:* many network functions rely on tables. Thus, dedicating more chip area for memory is desirable. E.g., the Reconfigurable Match Tables (RMT) switch [3] allocates over 50% of the chip area for memory. Switches use SRAM and TCAM. They have a typical access time of 0.5-2 nanoseconds, which is close to a switch clock cycle. More on-chip memory also reduces expensive data movement to off-chip memory.

- *Reduced data size and type:* implementations use 8-bit to 32-bit words [3]. Such narrower and simpler data types suffice for header field operations and can be manipulated with simple ALUs. Since ALUs are simple, more units can be implemented into the chip. Reduced data size also increases the effective memory bandwidth and on-chip memory utilization.

- *Domain-specific programming language:* switches are programmed with P4. P4 instructions are limited to typical operations applied to header fields. P4 is protocol-independent (the switch is not tight to any protocol) and

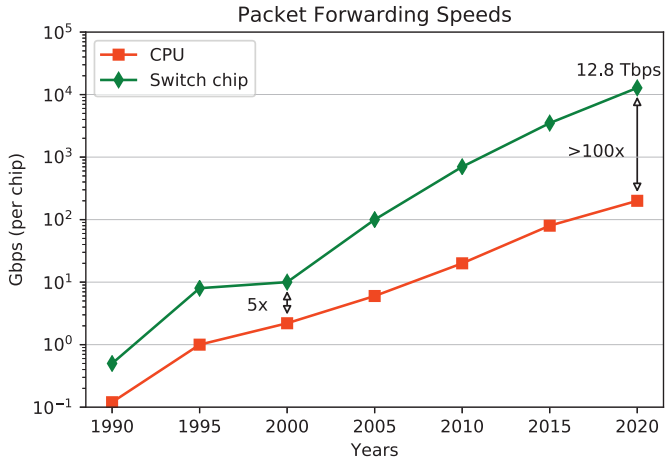

Figure 3: Evolution of the packet forwarding speeds of the general-purpose CPU and the switch chip (reproduced from [38]).

target-independent (the ASIC is hidden from the programmer). This programming abstraction makes porting applications flexible to a variety of targets.

- *Performance:* switch chips have remained faster at switching than CPUs for a long time and the gap is only increasing, as shown in Fig. 3. Currently, the fastest switch chip is over 100 times faster than the CPU.

The performance gain of switches relies on the multiple dimensions of parallelism, as described next.

- *Parallelism on different stages*: each stage of the pipeline processes one packet at a time [40]. In Fig. 2, the number of stages is N. Implementations may have more than 10 stages on the ingress and egress pipelines. For example, each pipeline in the Tofino architecture allows up to 12 stages on ingress and egress [41]. While adding more stages increases parallelism, they consume more area on the chip and increase power consumption and latency.

- *Parallelism within a stage*: the ASIC contains multiple match-action units per stage. During the match phase, tables can be used for parallel lookups. In Fig. 2, there are four matches (in blue) on each stage that can occur at the same time. Each header field is processed by an ALU (e.g., IP destination address, Media Access Control (MAC) source address, etc.) to execute one operation over the header field, enabling parallel actions on all fields. Hundreds of match-action units exist per stage and thousands in an entire pipeline [40]. Since ALUs execute simple operations and use a simple Reduced Instruction Set Computer (RISC)-type instruction set, they can be implemented in the silicon at a minimal cost.

- *Very Long Instruction Words*: the set of instructions issued in a given clock cycle can be seen as one large instruction with multiple operations, referred to as Very Long Instruction Word (VLIW). A VLIW is formed from the output of the match tables [37]. A stage executes one VLIW per packet, and each action unit within the stage executes one operation. Thus, for a given packet, one operation per field per stage is applied [3].

- *Parallelism on pipelines*: the switch chip may contain multiple pipelines per chip, also referred to as pipes. Pipes on a PSA device are analogous to cores on a general-purpose CPU. Examples include chips containing two and four pipes [37, 42]. Each pipe is isolated from the other and processes packets independently. Pipes may implement the same functionality or different functionalities. For a packet to traverse through two or more pipes, it must be resubmitted or recirculated. Packet resubmission and recirculation send the packet back for ingress processing. Packet resubmission occurs at the end of the ingress pipeline whereas packet recirculation occurs at the end of the egress pipeline, see Fig. 2.

Table 2 summarizes the features of programmable switches and contrasts them with general-purpose CPUs.

3.3. Advantages of Programmable Switches

Advantages of P4 programmable switches include:

- *Agility*: the programmer can design, test, and adopt new protocols and features in significantly shorter times (i.e., weeks rather than years) using the same switch hardware.

- *On-field programmability*: reprogrammability enables the programmer to add or modify features that were unforeseen at the time of deployment, saving time and cost and increasing security. For example, as new vulnerabilities are discovered after the switch is deployed, subsequent reprogrammability can be applied to the device.

- *Flexibility and customization*: the match process can be made over multiple header fields associated with different protocols at different layers (e.g., tuples). Similarly, the action process can be customized according to the programmer's needs, such as performing general and customized security-related functions (rewriting header fields, blocking/dropping a packet, sending a packet to a special server for DPI).

- *Visibility*: programmable switches provide greater visibility into the behavior of the network. Security-related functions can be improved by quickly detecting anomalies such as DDoS at nanosecond timeframes. On the other hand, detecting those anomalies with legacy methods (e.g., Netflow, sFlow) would take seconds at best. Moreover, events occurring at very low timeframes may not be detected by traditional methods.

- *Enhanced resource utilization and reliability*: fixed-function switches incorporate a large superset of features. They consume resources and add complexity to the processing logic, which is hard-coded in silicon. With programmable switches, the programmer has the option to

Table 2: Comparison Between Programmable Switches and General-purpose CPUs

Feature	Programmable Switch	General-purpose CPU	Notes
Packet forwarding speed	Up to 12.8 Tbps	≈ 200 Gbps	Switch enables faster detection of anomalies and security events
Memory	Limited fast memory (on-chip SRAM and TCAM)	Abundant slower memory (off-chip DRAM)	Switch has limited number of fast tables to store firewall entries, signatures, etc.
Data type and size	Reduced, simpler (bool, int, var-bit, etc. [35])	General, complex (8-bit word, 32-bit float, etc. [39])	Switch has simpler data types that affect several security implementations (e.g., no floating points to store measures such as entropy)
ALU	Simple 8-bit to 32-bit ALUs optimized for line-rate packet header operations	Complex computation units optimized for memory load and store operations	Switch operations are limited to simple arithmetic, logical, and bit manipulation; no encryption, regex processing, etc.
Language	P4	Java, Python, etc.	Level of abstraction for networking operations is raised with p4; compiler exploits inherent operation parallelism
Parallelism	Multiple pipelines, multiple stages per pipeline, multiple ALU and tables per stage	Multiple cores (multi- core microprocessors)	Switch has higher throughput for packet header operations (e.g., faster detection of attacks, multiple operations at a time, etc.)

implement only those features that are needed. Removing unused features minimizes the risk of software vulnerabilities and failures, increases the reliability of the device, and reduces power consumption. Furthermore, unused resources can be better utilized.

- *Top-down design*: for decades, the networking industry operated in a bottom-up approach. A fixed-function ASIC is at the bottom and enforces available protocols and features to the programmer at the top. With programmable switches, the programmer rather than the chip manufacturer describes protocols and features in the ASIC. The evolution of protocols and features is quicker and customized to the needs of the network owner.

- *Enhanced performance*: when security functionalities are offloaded to the switch ASIC, the performance is enhanced accordingly.

3.4. Limitations of Programmable Switches

Limitations of P4 programmable switches include:

- *ALU capability*: the amount and type of processing are limited by the simple instruction set, which is not intended for manipulation of the packet body, such as encryption or regular expressions [3].

- *On-chip memory capacity*: on-chip memory is expensive and limited in size. Switch chips incorporate a certain amount of TCAM and SRAM. Current programmable switches have capacities of approximately 20 megabytes [42]. This contrasts with general-purpose CPUs which have abundant off-chip memory.

- *Simple per-stage actions*: to preserve line-rate processing, only a specific number of simple operations are permitted [43].

- *Limited concurrent memory access*: packets can read or write to specific memory addresses. Thus, simple functions such as finding the minimum of k-heavy hitters in the data plane are challenging to implement [43].

- *Single stage memory access*: the RMT model allows access to the stateful memory only from one pipeline stage at a time to avoid read-write hazard [43].

- *Monolithic P4 program*: Despite the flexibility of the P4 language and its accommodation of a range of targets, it creates a tight coupling between P4 programs and the underlying architectures. Thus, P4 programs are often written with a flat top-level structure and a specific set of headers and metadata that are scattered throughout the whole program. Such a design hinders code reuse and makes minor modifications to the code result in altering the rest of it [44].

- *Architecture-specific P4 program*: P4 programs depend on the pipeline model of the architecture they are developed on. The available architectures differ in a number of functionalities. For instance, the Tofino model allows packet resubmission only once, whereas BMv2 allows infinite packet resubmission [23, 44].

4. Methodology and Taxonomy

4.1. Survey Methodology

The survey starts with a broad search scope to provide relevant information related to the history of data plane programmability. Then, it narrows the scope towards literature exclusive to P4 and more specifically to security techniques implemented in the data plane. In the later part, the survey describes the main P4 applications and pinpoint a number of security implications using the STRIDE model.

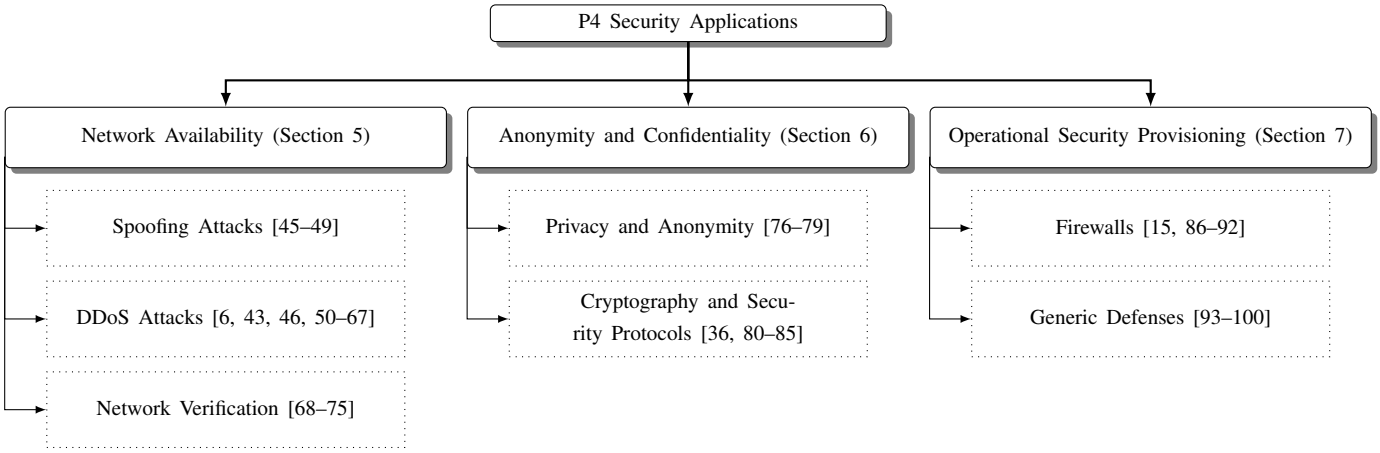

Figure 4: A security-centric taxonomy of programmable switches literature based upon relevant, explored research areas.

4.2. Taxonomy Overview

The proposed taxonomy of P4 security applications is shown in Fig. 4. It classifies applications based on network availability, anonymity and confidentiality, and operational security provisioning. Each subsection presents a brief background of the discussed topic (e.g., DDoS, firewall, etc.) followed by a literature review comprising of related P4 papers. Then the subsection dedicates two segments to discussions and comparisons. The first segment dives into implementation details of the discussed P4 papers and compares them against each other (intra-category comparison). The second segment compares the P4 applications with their traditional counterparts.

5. Network Availability

The first category in the taxonomy (Fig. 4) presents applications and techniques pertaining to network availability. In general, this category ensures that a network is available and working as expected. In this section, three main types of P4 applications related to network availability are addressed. First, the P4 applications that mitigate spoofing attacks. Second, those targeting DDoS attacks that hinder the availability of the network and disrupt its operations. Finally, network verification techniques that aim to verify the correctness of the P4 program, thus, avoiding faulty programs that would undoubtedly affect the network's availability.

5.1. Spoofing Attacks

5.1.1. Background

A spoofing attack occurs when a malicious party impersonates another device for the purpose of launching attacks (e.g., DDoS) that can render the network unavailable, stealing data, and disseminating malware, or bypassing access controls. For instance, CloudFlare [101] reports that all layer-3 gigantic DDoS attacks require spoofing. Common types of spoofing are IP address spoofing attacks, Address Resolution Protocol (ARP) spoofing attacks (modified MAC address), and Domain Name Server (DNS) server spoofing attacks (DNS server is modified to redirect specific domains to a different IP address). DNS server spoofing can be realized via Dynamic Host Configuration Protocol (DHCP) spoofing, where the attacker can modify the default gateway and the DNS server of DHCP clients [48].

Spoofing attacks remain a major threat in the network as existing anti-spoofing defenses (e.g., host-based approaches) cannot cope with the increasing attack rates. For instance, reports from CAIDA show that 30,000 spoofing attacks occur per day [102]. Recently, programmable switches are realized to prevent spoofing attacks by utilizing stateful memory processing, where switches can store information about hosts and detect spoofing attempts on the fly.

5.1.2. Literature

Li et al. [45] propose NetHCF, a line rate in-network system for filtering spoofed traffic based on the Hop Count Filtering (HCF) approach. NetHCF maintains a mapping table between the IP addresses and the number of hops (IP-to-Hop-Count (IP2HC) mapping table) a packet takes while traversing

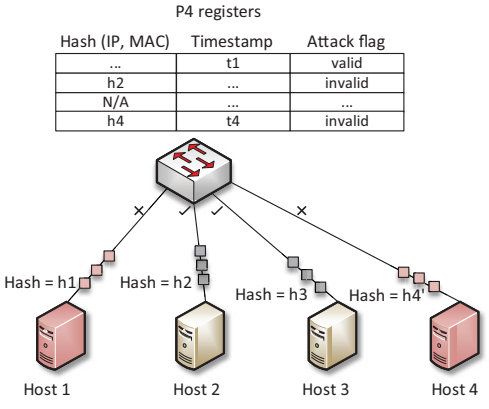

Figure 5: DroPPPP workflow [46]. Traffic from host 1 is denied since the flag is valid and the time difference is less than 1 s. Traffic from host 2 is allowed since the hashes match and the flag invalid. Traffic from host 3 is allowed since there are no stored hashes (subsequently, the entry gets updated after the first occurrence). Traffic from host h4 is denied since the hashes do not match and the time difference is less than 5s.

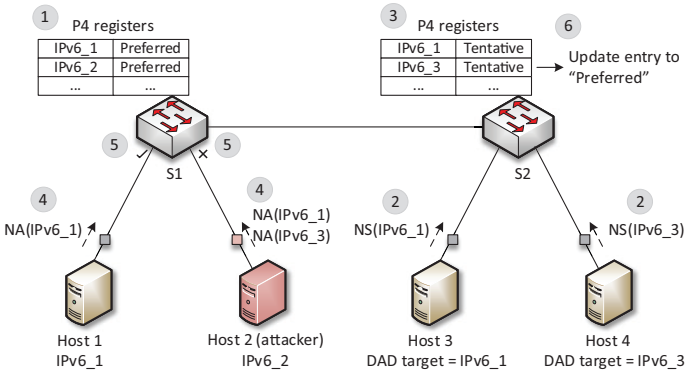

Figure 6: P4DAD workflow. 1- Switch S1 binds the entries IPv6_1 and IPv6_2 of hosts h1 and h2. 2- Hosts 3 and 4 send Neighbor Solicitation (NS) messages to be assigned an IPv6 address. 3- Switch S2 stores the corresponding IPs in its registers as tentative. 4- Host 1 sends a Neighbor Advertisement (NA) to point that it is assigned IPv6_1, and host h2 (attacker) sends an NA messages with IPv6_1 and IPv6_3. 5- Switch S1 will only permit non-spoofed NA messages (from host h1) based on the mapping table. 6- Switch s2 will remove IPv6_1 entry, and set IPv6_3 to permanent.

the network. The IP2HC table can be used to infer spoofed IP traffic by comparing the number of hopes in the table with that belonging to spoofed traffic. Such an approach is effective assuming that the table maintains hop-count values of legitimate traffic, and the attacker cannot easily use the correct hop-count values in the attack traffic.

DroPPPP [46] is an approach that detects spoofed addresses by hashing the source IP and MAC addresses of an incoming packet and matching it against the old hash values stored in the switch. Fig. 5 illustrates how DroPPPP performs packet filtering to detect spoofing based on different scenarios.

Duplicate Address Detection (DAD) is used in the process of IPv6 address configuration, where nodes verify if an IPv6 address conflicts with another node. Since DAD messages are not authenticated or encrypted, an attacker can launch a spoofing attack against nodes in the IPv6 network. Motivated by such a vulnerability, Kuang et al. [47] propose P4DAD to secure DAD against spoofing attacks by storing bindings between the IPv6 addresses and ports in the switch similar to DroPPPP's design [46]. Fig. 6 depicts the workflow of P4DAD.

Narayanan et al. [48] implement commonly used defense mechanisms against spoofing attacks using the P4 language. The proposed approach is based on match-action table rules initially inserted by the controller (or statically hardcoded) to specify trusted users. Gondaliya et al. [49] explore different implementation techniques of anti-spoofing mechanisms using programmable switches. Namely, the anti-spoofing mechanisms implemented are: Network Ingress Filtering (NIF) [103]; Loose Reverse Path Forwarding (RPF-Loose) [104]; Strict Reverse Path Forwarding (RPF-Strict) [104]; Feasible Path Reverse Path Forwarding (RPF-Feasible) [104]; Spoofing Prevention Method (SPM) [105]; and Source Address Validation Improvement (SAVI) solution for DHCP [106].

5.1.3. *Anti-spoofing Defenses: Comparison, Discussions, and Limitations*

Table 3 summarizes and compares the aforementioned address spoofing implementations in P4. NetHCF stores a small portion of the IP2HC table in the data plane and the entire table in the controller. IP2HC entries in the data plane correspond to the most active IPs in the network. NetHCF utilizes the available memory on the switch by storing the IP2HC mapping table in a binary tree format using registers (tree nodes with a common IP prefix and hop-count value are aggregated into one node). Furthermore, NetHCF captures legitimate hop-count changes at line rate (using P4 registers and bitmaps) and uses message digests (IP address, Time To Live (TTL), and TCP flag) instead of the whole packet to reduce the communication cost between the control and data planes.

DroPPPP [46] and P4DAD [47] store heuristics for every port connected to a host. In particular, DroPPPP uses three registers

Table 3: Address Spoofing Implementations in P4

Paper	Attack Vector			Anti-spoofing Technique	Control-Data Interaction	Stateful Memory	Target	Limitations
	IP	ARP	DHCP					
NetHCF [45]	✓			IP-to-Hop-Count mapping table	Medium	Rule table	Tofino	Vulnerable to saturation attacks on the controller
DroPPPP [46]	✓	✓		Matching packet hash with old heuristics	Low	Rule table, src IP, MAC hash, time-stamp, flag	BMv2	Requires full deployment for high detection rate
P4DAD [47]	✓			Binding between IPv6 addresses/ports	Low	Rule table, IPv6, states	BMv2	Might be inscalable for large number of hosts
[48]	✓	✓	✓	DHCP snooping, IP source guards, MAC source guards	Low	Rule table	BMv2 NetFPGA	Requires maintenance of entry tables from the administrator
[49]	✓		✓	NIF, RPF-Loose, RPF-Strict, RPF-Feasible, SPM, SAVI	Low	Rule table	NetFPGA	Requires maintenance of entry tables from the administrator

Table 4: P4 and Traditional Anti-spoofing Defenses

Feature	P4-based	Traditional Anti-Spoofing	
		Host-based	Router-based
Performance	High (intelligence is on the switch)	Low (attacks reach the hosts)	High (intelligence is on the router)
Latency	Low	High	Low
Visibility	High (network-wide)	Low per-host)	High (network-wide)
Flexibility	Medium (limited to P4 and the available resources)	High (easily modified; abundant in resources)	Low (requires hardware modification that takes years)

to store a hash value of the source IP and MAC addresses of the host, a timestamp for the last detected attack packet, and a flag that is valid if an attack is ongoing (attack flag). DroPPPP workflow is depicted in Fig. 5. On the other hand, P4DAD's registers store the IPv6 addresses (only the 64-bit interface identifier) and their corresponding states (e.g., preferred, tentative).

[48] and [49] implement the different anti-spoofing techniques using match-action tables. Evaluations in [49] show that SPM consumes the highest number of resources among all anti-spoofing techniques operating at the network layer. Such an observation is due to the fact that SPM requires an extra match-action table compared to the other schemes.

5.1.4. Comparison with Traditional IP Spoofing defenses

Table 4 compares P4-based anti-spoofing defenses with their traditional counterparts. Traditional source spoofing defenses are categorized into router-based [107–109] and host-based [110–112] defenses. Router-based anti-spoofing installs the defense mechanism into the router to trace the source of the attack and blocks the malicious traffic. Such an approach might require vendors to modify the hardware to support the anti-spoofing mechanism. Host-based anti-spoofing installs the defense mechanism on the end system, thus making it easier in terms of deployment. However, host-based defenses use complex source-discrimination schemes [113–115], or reduce the resource consumption of each request [116, 117] to mitigate attacks. P4-based anti-spoofing approaches combine the performance gain in traditional router-based defenses, with the ease of deployment (programmability) in traditional host-based defenses.

5.2. DDoS Attacks

5.2.1. Background

DDoS attacks have been persistent as a large scale threat among network attacks for a long time. Throughout the years, DDoS attacks have evolved to become more diverse and complicated to cope with, leading to more than 400,000 DDoS attacks with a peak volume topping terabits. Emerging programmable switches can facilitate the development of robust techniques for DDoS defense while maintaining high throughput and accuracy [6]. The specifications of such programmable devices and their adoption to stateful packet processing allow them to process packets at terabits line rate. Furthermore, programmable switches proved to overcome several limitations of traditional heavy hitter detectors (e.g., packet sampling [118]) that have limited accuracy since they are implemented on general-purpose CPUs. Heavy hitters are flows with large traffic volumes measured in terms of the number of packets, bytes, connections, etc. [119]. Heavy hitters constitute most of the network traffic over a period of time, and they can be used for malicious purposes such as volumetric DDoS attacks.

5.2.2. Literature

Sivaraman et al. [50] propose HashPipe, a heavy hitter detection algorithm that identifies the k-heaviest flows with high accuracy entirely in the data plane. Probabilistic RECirculation admisSION (PRECISION) [43] is a heavy hitter algorithm that recirculates a small fraction of packets for a second pipeline traversal. MV-Sketch [51] is an invertible sketch that applies the majority vote algorithm [120] to find candidate heavy flows, i.e., flows having a high likelihood of carrying the largest amount of traffic among all the flows. Kucer et al. [52] present another line of research that targets hierarchical heavy hitters, changes in traffic patterns, and superspreaders. Hierarchical heavy hitters (hierarchical aggregates) extend the notion of frequent items to data arranged in a hierarchy. Aggregation can be in the form of source/destination IP address and port, protocol fields, etc. The developed approach, namely Elastic Trie, can perform security tasks, such as DDoS, anomaly, worm, and spam detection. Harrison et al. [53] focus on detecting network-wide heavy hitters to mitigate attacks that cannot be detected when monitored at a single switch, such as port scanners and superspreaders/DDoS. Ding et al. [54] build on top of [53] and propose an approach for incrementally deploying programmable switches in legacy infrastructure for monitoring as many distinct flows as possible. The authors also propose a network-wide heavy hitter detector that is compatible with the incremental deployment approach.

As for schemes that solely focus on DDoS detection, Zhang et al. [6] propose POSEIDON, a notable approach against various volumetric attacks. The high-level architecture is depicted in Fig. 7. Essentially, POSEIDON provides network operators with a simple and modular expressive policy language to specify their DDoS attack mitigation. POSEIDON partitions the needed functions across the available P4 switches and general-purpose servers. Furthermore, POSEIDON can adapt to dynamic attacks by generating a new defense policy and reconfiguring the switch with a new P4 program through the intervention of the controller and temporary servers. POSEIDON relies on flow-level states instead of per-packet information to detect DDoS. This approach could be easily evaded in stateful protocols-based DDoS traffic. Accordingly, the authors extend their work in [55] to include a Finite State Machine (FSM)-based monitor, which can capture these stateful packet processing behavior. In contrast to POSEIDON, Jaqen [56] is a switch-native DDoS detection and mitigation approach that runs entirely in the data plane without relying on external hardware. In particular, the approach covers a broad spectrum of volumetric

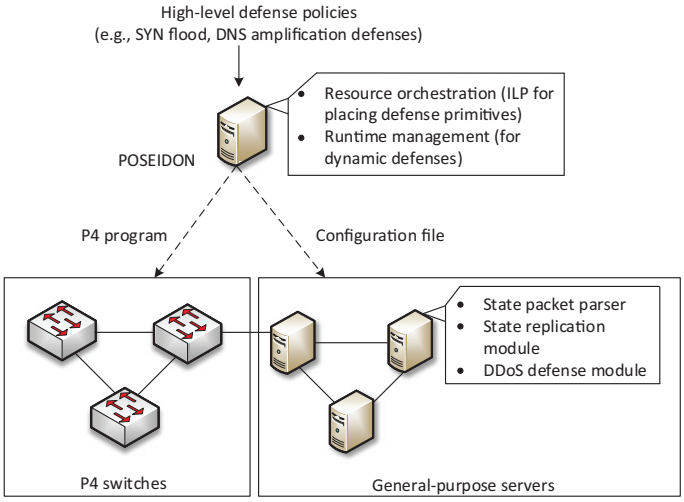

Figure 7: POSEIDON high-level architecture. POSEIDON runs on an external server to take the policies defined by the operator, configure, and orchestrate the underlying data plane. The latter consists of P4 switches and general-purpose servers that assist the switches in complex operations.

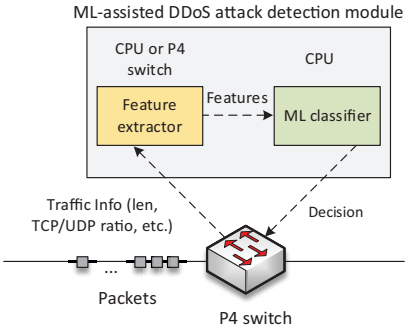

Figure 8: Workflow of [62] depicting the interaction between the switch and the DDoS detection module.

DDoS attacks within Internet Service Providers (ISPs) without interrupting client-side servers. Furthermore, Jaqen provides an API to obtain metrics for detection (e.g., querying User Datagram Protocol (UDP) heavy flows), as well as a flexible mitigation API that can be used to cover other types of attacks.

Lapolli et al. [57] overcome the limitations of standardized monitoring techniques, such as packet sampling, which incur substantial overheads on the network performance, by designing a fully in-network inspection mechanism against DDoS attacks. The detection mechanism, namely DDoSD-P4, encompasses three stages, namely, entropy estimation [121], traffic characterization, and anomaly detection. The authors of DDoSD-P4 extend their work to develop EUCLID [58], which enables scalable detection and mitigation of volumetric DDoS attacks entirely in the data plane. Dimolianis et al. [59] combine three essential traffic metrics of DDoS attacks discussed as follows. (1) Total number of incoming traffic flows within an interval. (2) Percentage of flows directed towards a network out of the total incoming flows, referred to as the significance of a network. (3) Symmetry ratio of incoming to outgoing packets. Furthermore, the approach can pinpoint the victim's destination by monitoring incoming and outgoing flows of a specific network. Ding et al. [60] design BACON, a probabilistic data structure that combines Bitmaps [122] and Count-Min Sketch (CMS) [123]. In contrary to entropy-based DDoS detection, which can only detect attacks, BACON is cardinality-based that can estimate per-destination flow cardinality and identify the victim. Furthermore, the authors propose INDDoS, an approach that employs BACON data structure entirely in the data plane to perform DDoS detection with victim identification.

Friday et al. [61] propose a unified in-network detection and mitigation approach against a broad spectrum of attacks, mainly volumetric and stealthy DDoS attacks. The detection is based on statistics (e.g., flow count, timestamp) collected from the P4 switch. Musumeci et al. [62] use Machine Learning (ML) classification techniques to perform TCP flood detection from a set of features (average size of packets, TCP/UDP ratio, etc.) extracted from P4 registers periodically. The detection framework is mainly composed of two components, the P4 switch, and the ML module. The former receives the packets and forwards them to the latter that indicates the legitimacy of the traffic, see Fig. 8.

Paolucci et al. [63] explore P4 deployment in an SDN multilayer packet-over-optical networks, specifically at the edge where a plethora of traffic flows traverse. The approach showcases the adoption of P4 in two scenarios, namely traffic engineering and SYNchronize (SYN) flooding based on header inspection and packet counting. Essentially, the SYN flooding defense includes a customized parser (up to layer-4) and a stateful memory to store the number of attempts matching the TCP SYN flooding behavior.

Scholz et al. [64] leverage programmable switches to protect a multitude of servers against SYN flooding attacks, i.e., the switch is used as SYN proxy. In particular, the authors implement two strategies against SYN flooding attacks, namely, SYN cookies and SYN authentication. In SYN cookie, instead of storing the state of the connection (timestamp, maximum segment size, hash of 4-tuple), the state is encoded and sent as the sequence number by the client. The SYN proxy (characterized by the P4 switch), in turns, decodes and verifies (cookie calculation) the sequence number before establishing the connection. In SYN authentication, the SYN proxy expects a certain response to a triggered event (whitelisting).

Simsek et al. [46] propose DroPPPP, a P4-based approach to detect and mitigate spoofed-based DoS attacks. The IP spoofing mitigation is summarized in Section 5.1.2. In the DoS detection, the authors employ Two Rate Three Color Marker [124] to identify and block hosts performing the attack. Two Rate Three Color Marker compares the rate of incoming packets to a certain threshold and acts accordingly.

Kuka et al. [65] present an approach that defends against amplification attacks. Conceptually, the operator specifies a set of rules to monitor a subset of traffic (e.g., based on destination IP address) and the traffic limit. Subsequently, the data plane sends statistics (stored outside the P4 pipeline) related to the controller, which analyzes the traffic and inserts rules to block malicious activities. Similarly, the authors in [66] propose DIDA,

a distributed in-network defense architecture against Amplified Reflection DDoS (AR-DDoS) attacks, also referred to as DNS reflection/amplification, using P4. The authors implement their defense strategy entirely in the data plane without relying on any third-party tools and with the mere involvement of the controller. To make DIDA scalable and efficient, the authors count the responses at the border routers, while the requests are monitored at the access routers. Once an attack is detected, the border router installs Access Control Lists (ACLs) in its flow table to drop future packets from the corresponding IP address.

Contrary to [66], Febro et al. [67] present a first-hop DDoS detection and mitigation, i.e., only the switch nearest to the attack source performs traffic counting and filtering. The proposed approach targets Session Initiation Protocol (SIP) DDoS attacks based on the number of SIP INVITE and REGISTER packets.

5.2.3. *DDoS Defenses: Comparison, Discussions, and Limitations*

Table 5 summarizes and compares the aforementioned P4-based DDoS attack mitigation techniques. HashPipe is adapted from a strawman solution, HashParallel [50], that requires recirculating every unmatched packet. HashPipe enhances the accuracy at the expense of the throughput, while HashParallel sacrifices the accuracy to achieve higher throughput. In contrast to HashPipe and HashParallel, PRECISION [43] recirculates a small fraction of packets to conform to the memory limitations of the programmable switch while achieving high accuracy and performance. The invertible data structure MV-Sketch [51] is shown to achieve higher accuracy and smaller resource usage than PRECISION using the same CAIDA dataset. However, PRECISION has a lower relative error than MV-Sketch. As for network-wide heavy hitters, the authors in [54] show that the work by Harrison et al. [53] has low accuracy when a legacy switch exists in the network and a switch is more likely to fail in detecting heavy hitters when it receives few distinct flows. As a result, the proposed approach in [54] uses HyperLogLog [125] to estimate the number of distinct flows. Evaluations show that the network-wide heavy hitter detection in [54] outperforms the one presented in [53] while incurring additional packet processing time.

The main advantage of Jaqen [56] over POSEIDON [6] is its ability to fully work in the switch, thus reducing the costs of external servers and achieving line rate speed. In particular, the authors utilize universal sketches [126] for network monitoring since they can estimate a range of network statistics from multiple algorithms, such as heavy hitters, distinct flows, and entropy. To combat the hardware constraints in switches, Jaqen assumes that the ISP encompasses several hardware resources and develops a network-wide DDoS detection. Jaqen has a broader detection coverage than POSEIDON, such as identifying anomalies using entropy, handling distinct TCP/UDP flows, etc.

[57], [59], and [60] implement their DDoS defense in the data plane without relying on external resources. For entropy estimation in [57], the authors use count sketch data structure (implemented via P4 registers) to approximate the frequencies of

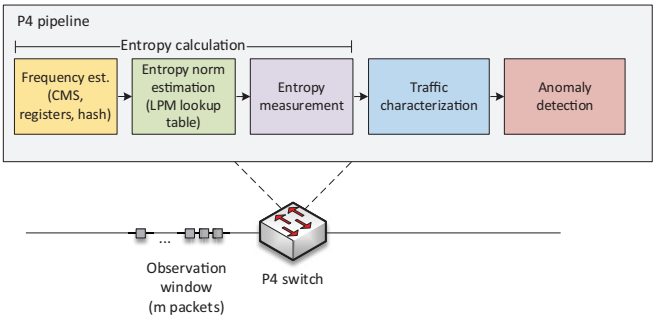

Figure 9: Entropy estimation in the data plane [57].

IP addresses. Furthermore, binary logarithmic values are precalculated using Longest Prefix Match (LPM) lookup tables implemented in TCAMs, see Fig. 9. Results show that the entropy estimation error fluctuates between 5% and 1%, where higher accuracy means higher processing, e.g., more hash functions processing. In [59], the monitored networks are initially specified as match-action table rules. Registers hold statistics, such as flow counters, and the storing of values is based on Bloom Filter (BF) data structure. Floating point operations are realized by multiplication and bitwise shifting operations (for division). INDDoS [60] uses BACON data structure to identify destination IPs (victims) that are targeted by source IPs exceeding a dynamic threshold. INDDoS can be considered as complementary to approaches that assume the DDoS victim is known in advance (e.g., POSEIDON).

To mitigate volumetric attacks, the approach in [61] first hashes the SYN packet's fields (window size, TTL, IP options length, etc.) and the index is used to access P4 counters and increment the value representing the number of SYN requests for a specific flow. Likewise, stealthy attacks are mitigated by hashing the source IP address and storing the timestamps. Likewise, [62] relies on statistics extracted from the packet's header to feed an ML classifier that detects SYN flooding attacks. In the feature extraction process, evaluations show that the P4 switch is capable of extracting the packets' header and metadata significantly faster than the deployed server.

In [64], initially, the P4 program parses up to the TCP header. Since P4 does not support cryptographic hash functions (e.g., SHA1, SHA2), the authors use SipHash function [127] implemented as a P4 extern for efficiency. After a client establishes the TCP handshake with the SYN proxy, it can immediately start sending data. Meanwhile, the SYN proxy could be still establishing a TCP handshake with the server; thus, the sent data could be dropped. To cope with such a limitation, the P4 switch forwards the packets to an external storage server for packet buffering until the TCP handshake is established (with the involvement of the controller). The workflow of [64] is depicted in Fig. 10.

[65] modifies the parser to match on rules specified by the operator, however, they did not utilize the stateful memory processing on the switch to perform line rate attack detection (the controller analyzes and insert the rules).

In [46], the authors use P4 meters to implement the two rate

Table 5: DDoS Defenses Schemes in P4

Paper	Attack Vector	P4-specific Implementation	Data Structure	Rules Based on	Target	Limitations
HashPipe [50]	Volumetric	Heavy hitter detection using a pipeline of hashes	Hash tables	5-tuple	BMv2	Sacrifices accuracy for performance
PRECISION [43]	Volumetric	Heavy hitter detection using probabilistic recirculation	Hash tables	IP src, dst	Tofino	Recirculation may impact the accuracy
MV-Sketch [51]	Volumetric	Heavy hitter detection using invertible sketches	Invertible sketches	5-tuple	Tofino	Predefined (static) threshold, requires controller interaction
Elastic Trie [52]	Volumetric	Hierarchical heavy hitter detection	Elastic Trie	IP src	BMv2	Predefined (static) threshold, requires controller interaction
[53]	Volumetric	Network-wide heavy hitter detection	N/A	IP src	Tofino	High memory overhead due to per-key state storing
[54]	Volumetric	Incremental deployment for network-wide heavy hitters	CMS	IP src, dst	BMv2	Cannot handle adding new switches (only replacement)
POSEIDON [6, 55]	Volumetric*	Expressible DDoS defense policies	CMS	Generic (ICMP, TCP, HTTP, etc.)	Tofino	Spoofed IP addresses can overwhelm switch's memory
Jaqen [56]	Volumetric*	In-line detection and network-wide coordination	Universal sketches	Generic (ICMP, TCP, HTTP, etc.)	Tofino	Requires few seconds to react to attacks
DDoSD-P4 [57]	Volumetric	Shannon entropy (detection)	Count sketch	IP src, dst	BMv2	No granular precision, limited attack coverage
EUCLID [58]	Volumetric	Shannon entropy (detection and mitigation)	Count sketch	IP src, dst	Tofino	No granular precision, limited attack coverage
[59]	Volumetric	(1) Traffic feature (2) Network significance (3) Symmetry ratio	BF	5-tuple hash	Netronome	Limited measurements, deteriorating performance for high packet rates
INDDoS [60]	Volumetric	Threshold-based victim detection	BACON	IP src	Tofino	Consumes all the available pipeline stages
[61]	SYN flooding / stealthy	Packet and timestamp counting (on signature matching)	BF	SYN header (e.g., IP options len.), IP src hash	BMv2	Hash collisions are not addressed
[62]	SYN flooding	ML on features from packet header (e.g., packet length)	N/A	IP, UDP, TCP	BMv2	Latency incurred by the ML decision
[63]	SYN flooding	Packet counting	N/A	IP, TCP	BMv2, NetFPGA-S	Manual configuration of threshold
[64]	SYN flooding	SYN cookie, SYN auth.	N/A	TCP header fields	Netronome, NetFPGA-S BMv2, t4p4s	Extra latency incurred by the controller and the external storage server
DroPPPP [46]	Spoofed-based DoS	Two rate three color marker	N/A	MAC, IP src	BMv2	High CPU load under attacks and a poor choice of thresholds
[65]	Amplification	Extract header fields from affected traffic	N/A	IP, UDP, TCP	FPGA	No intelligence in the switch
DIDA [66]	Amplification	Request/response tracking	CMS , CHT	IP, port (src, dst)	BMv2	Hash collisions are not addressed
[67]	SIP DDoS	INVITE and REGISTER packets counting	N/A	SIP packets	Ethernet, IP, UDP, SIP	Saturation attack on the controller

Volumetric*: Can defend against a wide variety of volumetric attacks NetFPGA-S: NetFPGA-SUME

three color technique. On the other hand, [66] performs traffic monitoring entirely in the data plane by counting packets based on the CMS data structure and resetting the counters periodically. Furthermore, to maintain ACLs used in blocking attacks, the authors use Cuckoo Hash Tables (CHT) with four hash functions. Finally, [67] performs DPI to monitor and count INVITE

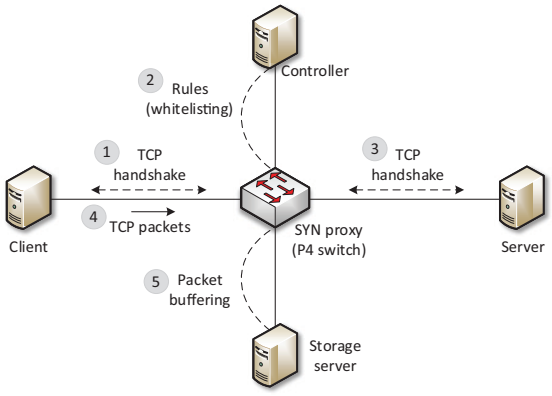

Figure 10: High-level architecture of [64]. 1- The client tries to establish a TCP connection with the P4 switch (SYN proxy). 2- The switch establishes the connection once validated via SYN cookie or SYN authentication using a controller. 3- The switch establishes a TCP connection with the server. 4- After the TCP session is established, the client starts sending packets. 5- Packets arriving from clients after the TCP handshake with the switch and before the TCP handshake with the server are buffered using the storage server.

and REGISTER packets on each port within the switch. Additionally, a controller periodically monitors the counters collected by the P4 switches. Based on the collected statistics, if the rate of SIP INVITE packets exceeds a threshold, the controller inserts rules to drop subsequent INVITE and REGISTER packets from the corresponding port.

5.2.4. Comparison with Traditional DDoS defenses

Table 6 shows a comparison between P4 and traditional DDoS defenses. Traditional defenses can be mainly categorized into two, traffic scrubbing centers, and SDN/Network Function Virtualization (NFV).

A traffic scrubbing center collects, analyzes, and mitigates malicious traffic. However, such an approach is expensive and proprietary, as it requires the deployment of a large cluster of commodity servers or proprietary middle-boxes. Furthermore, when new attacks emerge, upgrades to the middleboxes are often required to defend against these new variants, which usually takes time due to the compulsory negotiations between the vendor and the customer. These factors hinder the process of mitigating continuously evolving DDoS attacks [6].

SDN/NFV DDoS defenses provide a cheaper and centralized solution for DDoS detection and mitigation. Nonetheless, it is limited to OpenFlow, and saturation attacks on the controller could be exploited.

P4-based DDoS defense offer several orders of magnitude higher throughput than highly-optimized packet processing software [14, 16]; thus, making them perfect candidates against such attacks. However, resource constraints impose several challenges on DDoS defense implementations.

As for traditional heavy hitter approaches, they include packet sampling [118] that use general-purpose CPUs, thus, limiting the accuracy and performance. Advantages of P4-based heavy hitters include their ability to operate at line rate, as well as perform per-packet inspection rather than sampling-based heavy hitters.

Table 6: P4 and Traditional DDoS Defenses

Feature	P4 switch	Traffic Scrubbing Center	SDN/NFV
Cost	Cheap	Expensive	Cheap
Flexibility	High, agile	Low, proprietary	Medium (restricted to OpenFlow)
Latency	Low (ASIC speed)	High (general-purpose CPU)	High (control decision + rule insertion)
Memory	Low	High	Low
Per-packet counting	Granular	Limited	Limited
Statistical / policy based	Difficult	Easy	Easy (at the controller)
ML	Difficult	Requires custom-built appliance	Easy (at the controller)
DPI	Limited	Flexible	Restricted to Open-Flow

5.3. Network Verification

5.3.1. Background

Faulty data plane programs can wreak havoc in the network. For instance, faulty routers in two airline networks have grounded airplanes for days (for both Delta and Southwest Airlines), showing just how disruptive the effects of incorrect network behavior can be [69]. Thus, the practice of verifying the correctness of the data plane is inevitable to preserve the availability of the network.

Increased programmability has a plethora of advantages in the networking industry, yet it has introduced new challenges related to program verification and correctness. In this section, state-of-the-art network verification approaches that ensure the safety and correctness of P4 networks are surveyed. Such techniques can guarantee that the network behaves well under unprecedented scenarios, and is immune to various attacks.

5.3.2. Literature

Kheradmand et al. [68] propose P4K, an executable formal semantics of the P4 language using the K, a framework used to define programming languages using rules and configurations [128]. P4K supports multithreading (e.g., multiple threads of execution within a single P4 program) as well as network semantics (network topology, links, etc.), in which it analyzes a network of P4 programs rather than a single one.

Stoenescu et al. [69] develop Vera, a scalable solution for verifying and debugging P4 programs before and during runtime using symbolic execution. Based on program verification, Vera automatically unveils several bugs related to loops, parsing, deparsing, tunneling, overflows, and underflows within seconds. Furthermore, Vera verifies the correctness of a program by modeling its properties using Computation Tree Logic (CTL) [129] and checking them by running Vera on the P4 program.

Table 7: Verification techniques in the context of P4

Paper	Technique	What It Does	Applications / Use cases	Limitations
P4K [68]	(1) Symbolic model checker (2) Symbolic execution	(1) Define executable formal semantics to the P4 language (2) Symbolic model checker and deductive program verifier for P4	(1) Detect unportable program (2) Uncover P4 bugs (3) Verify P4 programs (4) State space exploration	No support for clone into egress
Vera [69]	Symbolic execution	(1) Scalable P4 program verifier (2) Guarantee bug-free program (3) Support multiple snapshot	Uncover P4 bugs	State explosion
bf4 [70]	Static verification	(1) Verify P4 programs (2) Generate controller assertions and propose fixes (3) Throws exceptions on errors	(1) Uncover P4 bugs (2) Fix P4 programs	(1) Partial support of the primitives (2) Static verification bounded to safety rules
p4v [71]	Z3 theorem prover on GCL	(1) Verify P4 programs (2) Define control plane interfaces to constrain data plane behavior	(1) Verify header validity (2) Verify round tripping (3) Uncover P4 program bugs	Cumbersome process of writing the control plane interfaces
ASSERT-P4 [72]	(1) Assertion checking (2) Symbolic execution	(1) Verify P4 programs (2) Expressive assertion language to express properties	Uncover P4 program bugs	State explosion
Gauntlet [73]	(1) Z3 SMT solver (2)Symbolic execution	Find bugs in P4 compilers using: (1) Translation validation (2) Symbolic execution	Uncover P4 compiler bugs	(1) Partial finding of bugs (2) Inefficient for large P4 programs
p4pktgen [74]	Symbolic execution SMT solver	Generate test cases for P4 programs automatically (packets, table entries, and expected paths)	Uncover P4 program bugs	Limited support of P4 features (e.g., header stack, header unions, hashes)
P4RL [75]	DDQN	(1) Query language for specifying properties (2) Verifying P4 programs using reinforcement learning-fuzz testing	(1) Validate P4 program's behavior (using p4q) (2) Uncover program bugs	Only models device-independent queries

Dumitrescu et al. [70] present a novel verification approach to ensure a bug-free P4 program (bf4) for a class of bugs at any given configuration. To guarantee a bug-free program, bf4 is comprised of two components that run at compile-time and runtime to check the P4 program and the inserted rules. Essentially, bf4 differs from other existing approaches in the sense that it does not stop after discovering bugs. Instead, it generates predicates that eliminate these bugs and fixes the P4 program when predicates cannot be generated with existing keys.

Liu et al. [71] develop p4v, a powerful scalable language-based verification tool for P4 that aims to decrease the prevalence of bugs during development. p4v defines a control plane interface that limits the behavior of the data plane using symbolic constraints.

Freire et al. [72] develop a data plane verification approach called ASSERT-P4, capable of model checking general security and correctness properties of P4 programs at compile-time. One of the checked properties, for instance, is to verify that dropped packets will not be forwarded eventually (can be used to evade resurrecting dead packets).

In a different approach, Gauntlet [73] tries to find as many bugs as possible in P4 compilers, rather than in P4 programs. The approach specializes to test front and mid-end compilers independent of the target, as well as it supports restricted forms of back-end testing. Gauntlet aims to find bugs in P4 compilers by generating random programs that trigger crashes in the compiler.

Nötzli et al. [74] develop p4pktgen, an approach for automatically generating test cases consisting of packets, table entries, and expected paths for P4 programs. To generate a packet, it has to follow the correct parser transitions, conditional branches, and table actions. Such requirements are translated into a set of Satisfiability Modulo Theory (SMT) constraints by symbolic execution of a given path.

Finally, P4RL [75] is a different scheme that uses a combination of fuzzing and reinforcement learning to automatically verify P4 switches at runtime. The authors define a high-level query language, namely, p4q, so that operators can specify the intended properties of the P4 program.

5.3.3. Verification Schemes: Comparison, Discussions, and Limitations

Table 7 summarizes and compares the aforementioned P4 verification schemes. Essentially, the aforementioned schemes differ in the way they approach the verification process (e.g., symbolic execution, static verification, etc.), in addition to the bugs they discover in existing P4 programs.

Programming languages in K need to be defined syntactically and semantically. In [68], the syntax is defined by converting the Backus-Naur form of P4 to K. As for the semantic, it is given using semantic rules over configurations holding the program and its context.

Vera [69] and bf4 [70] verify P4 programs at compile-time and runtime. On one hand, Vera translates P4 to SEFL [130] (network verification language), uses symbolic execution, and program modeling using CTL. On the other hand, bf4 uses static analysis.

p4v [71] and ASSERT-P4 [72] limit the behavior of the data plane by introducing constraints and assertions written by the designer. p4v defines the semantic of P4 by translating it to Guarded Command Language (GCL) and performs the validation on the generated GCL code. The adoption of GCL allows the approach to be portable to newer languages (e.g., $P4_{16}$). On the contrary, ASSERT-P4 translates the P4 program into C-based model and uses a symbolic execution engine to explore all paths searching for assertion failures.

Gauntlet [73] and p4pktgen [74] follow the same rationale in uncovering bugs in the data plane by generating test cases against the tested target. In particular, Gauntlet grows an abstract syntax tree from the P4 compiler, i.e., expanding branches of the tree at random, then converts it into a P4 program. p4pktgen generates custom packets for specific P4 programs using SMT constraints and solver. The SMT constraints are used to guarantee that the crafted packet follows the correct parser transitions. Subsequently, the SMT solver is used in the process of generating packets.

5.3.4. Comparison with Traditional Verification Approaches

Traditional data plane verification techniques reactively check network-wide properties by analyzing snapshots of the current data plane configuration. Techniques such as VERMONT [131], Anteater [132], Hassel [133], VeriFlow [134] mainly verify properties related to host reachability, isolation, blackholes, and loop-freedom.

Furthermore, traditional network verification techniques rely on brute force techniques to generate dozens of input packets and check whether the network device produces the expected output. Although this procedure is expensive to be performed, it is widely adopted in conventional closed-source devices as it is done only once at manufacturing. However, the prosperity of flexible programmable data planes has rendered this technique ineffective as the capabilities of the device are not fixed but rather customized by the programmer.

5.4. Summary and Lessons Learned

Spoofing: While anti-spoofing defenses that rely on the controller to keep a full view of information (e.g., IP2HC filtering in [45]) might be more scalable than switch-only defenses, they incur more overhead and are vulnerable to attacks on the controller (e.g., saturation attacks). Basic router-level filtering (ingress/egress filtering, RPF) can protect against all spoofing attacks only if this mechanism is deployed on all the routers. Distributed anti-spoofing approaches, such as SPM, have high detection efficacy but require periodic key distribution and more switch resources.

DDoS: Existing heavy hitter detectors in the data plane include schemes that work separately at each switch (e.g., [43, 50, 51]), as well as network-wide approaches (e.g., ([53]). The former approach detects heavy hitters at each switch, then pushes the results to the controller for further analysis. On the other hand, the latter approach includes coordination between multiple switches. While network-wide heavy hitter detection can be more efficient in terms of the switch's resources compared to per-switch detectors, they can incur more communication overhead in the network. Additionally, network-wide detectors need to consider the existence of hybrid networks containing legacy switches.

The majority of DDoS detection approaches focus on volumetric attacks (e.g., [6, 56, 57]). While some research work uses external hardware to handle computation-heavy tasks, such as in POSEIDON [6], a notable recent work, Jaqen [56], proves that it is feasible to detect a wide range of attacks only using programmable switches. Approaches that do not rely on external hardware are advantageous to infrastructures since they entail low capital costs and provide line rate performance. Additionally, a common practice among notable DDoS detectors is the interface they provide to the operator for analyzing, managing, and extending the detection process.

Other DDoS detection approaches focus on utilizing existing data structures (e.g., sketches), as well as designing new ones (e.g., BACON) to fit the whole mechanism in each switch without the need for network-wide coordination nor external hardware. As for application layer DDoS attacks, [67] parses the top layer in the Open Systems Interconnection (OSI) model to detect SIP DDoS. Since a select few P4-based DDoS detectors involve parsing beyond the traditional fields, this opens new horizons for future endeavors.

Network verification: In network verification schemes, traditional techniques are insufficient for programmable networks due to their dynamic and flexible behavior. Current existing approaches focus on verifying P4 programs and compilers to find the maximum number of bugs possible using techniques such as symbolic model checker, static and dynamic verification, SMT solver, etc. While symbolic execution is widely adopted, it suffers from state explosion when handling complex P4 programs. Another important property in network verification is automation since it reduces the burden on operators. Finally, it is crucial that the developed approaches are generic for all admissible topologies and network events.

6. Anonymity and Confidentiality

The second category in the taxonomy (Fig. 4) discusses anonymity and confidentiality solutions in P4. The first section discusses privacy and anonymity implementations to hide and/or obfuscate necessary information. The second section discusses cryptography and security protocols that can achieve confidentiality.

6.1. Privacy and Anonymity

6.1.1. Background

Anonymization tools can be mainly classified into two categories, offline and online. Offline anonymization tools collect

and store network traffic and perform the anonymization technique offline. On the other hand, online anonymization is performed on the fly while the traffic is traversing the network. Anonymization can be achieved in several ways, for instance, source information hiding involves the rewriting of the source address of the packet. Several fields in the packet's header can be modified as well, such as the TTL, IP identification field, and TCP initial sequence number [77].

6.1.2. Literature

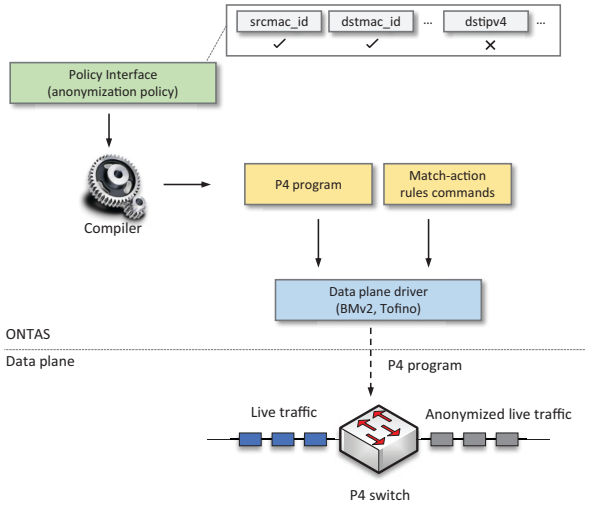

Figure 11: ONTAS [76] compiles the operator-defined policy into a P4 program compatible with the target, then pushes it to the underlying P4 switch.

Kim et al. [76] present an Online Network Traffic Anonymization System (ONTAS) by leveraging the capabilities of the programmable switches. ONTAS enables anonymization for packet fields independently. The network operator can choose whether to anonymize multicast and broadcast packets, Ethernet addresses, source and destination IP addresses, target MAC and IP addresses in ARP packets, etc.

Moghaddam et al. [77] introduce Practical Anonymity at the NEtwork Level (PANEL), a lightweight solution that enhances anonymization solutions in online communications between Internet users. Similarly, Datta et al. [78] propose Surveillance Protection in the Network Elements (SPINE) to obscure IP addresses and relevant TCP fields from the intermediate Autonomous Systems (ASes). SPINE only requires two trusted ASes (e.g., edge nodes) and host interference is not needed.

Another line of research focuses on obfuscating DNS traffic. Wang et al. [79] propose Programmable In-Network Obfuscation of Traffic (PINOT), a packet header obfuscation mechanism that obfuscates the association between client IP addresses and DNS requests, without modifying the DNS protocol, nor requiring any host-based interaction.

6.1.3. Privacy and Anonymity Schemes: Comparison, Discussions, and Limitations

Table 8 summarizes the aforementioned anonymization schemes. ONTAS [76] obfuscates Personally Identifiable Information (PII) addresses before sending them to a collector.

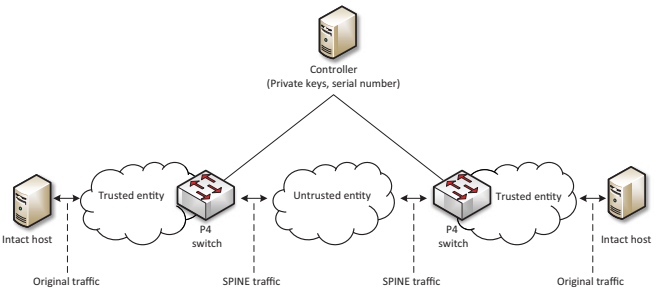

Figure 12: In SPINE [78], the IP addresses are encrypted before entering the intermediate ASes and decrypted when the packet exits the intermediate ASes.

The high-level architecture of ONTAS is depicted in Fig. 11. ONTAS supports prefix-preserving information for source and destination IP addresses, i.e., given an IP address, the prefix remains intact, and the remaining bits are obfuscated. Since anonymization results in changing the packet's header, and perhaps the packet's forwarding behavior, ONTAS applies a match-action table that clones the original packet and updates the packet's header before sending it to the egress port. Thus, updating the packet's field only for the cloned copy.

PANEL and SPINE ensured the anonymity of Internet users by manipulating certain header fields of the packet. PANEL [77] achieves anonymity, and session unlinkability. Anonymity is preserved using source address rewriting, source information hiding (randomized header fields), and path information hiding (randomized TTL). Session unlinkability is achieved by replacing session identifiers (e.g., TCP ports) with a pseudorandom number generator outputted by the local CPU. To preserve Internet routing and be able to route the response back, PANEL switches store the original session information in match-action tables. An additional property of PANEL is the compatibility with other legacy anonymity systems.

On the other hand, SPINE encrypts the IP addresses before the packets enter the intermediary ASes, see Fig. 12. Additionally, it encrypts the TCP sequence and ACKnowledgment (ACK) numbers to prevent associating a group of packets to the same TCP flow (session unlinkability). SPINE generates a fresh randomly-generated bit string and a one-time pad for each packet; thus, future encrypted IP addresses cannot be recovered. The encryption process in SPINE includes transforming IPv4 headers into IPv6 headers when packets leave the trusted entity and restoring the IPv4 headers upon entering the trusted entity. Such an approach is used to ensure that the routing remains successful despite IP address encryption. SPINE uses a centralized controller to generate and update keys and push them to the data plane match-action tables.

Finally, PINOT runs at the border of a trusted network, where it obfuscates each packet separately with a probabilistic encryption scheme, namely, two-round Even-Mansour (2EM) [135]. Thus, achieving sender anonymity, packet unlinkability, low deployment barriers, and low performance overhead. Technically, PINOT converts the IPv4 packet to an IPv6 packet that holds the encrypted IPv4 address; hence, the approach is stateless as all the needed information (except the secret key) for

Table 8: Privacy and Anonymity Schemes in P4

Paper	Anonymization Technique	Hash / Crypto. Function	Target	Limitations
ONTAS [76]	Header fields hashing	CRC	Tofino	(1) Does not support TCP/UDP field anonymization (2) One policy at a time
PANEL [77]	Source information rewriting and normalization, and path information hiding	N/A	Tofino	(1) Extra overhead during session initiation (2) First AS must be trusted (3) Limited number of served sessions
SPINE [78]	Header fields encryption	XOR, SipHash	BMv2	Supports IPv4 only
PINOT [79]	Header fields encryption	2EM	Tofino	Requires controller interaction for key management

Table 9: P4 and Traditional Anonymity Schemes

Feature	P4 Schemes				Traditional Schemes		
	[76]	[77]	[78]	[79]	[136]	[137]	[138]
Online	✓	✓	✓	✓	×	✓	✓
Offline	×	×	×	×	✓	✓	×
Line Rate	✓	✓	✓	✓	×	×	×
TCP/UDP	×	✓	✓	×	✓	✓	✓
Advanced Privacy	×	×	×	×	✓	×	✓
Controller Interaction	×	×	✓	✓	×	×	×

encryption/decryption is stored in the packet's header. A controller is involved in the approach in order to distribute the network encryption key. PINOT can encrypt the IP addresses in a single pipeline pass, which is a performance gain over other approaches such as SPINE that requires at least three passes.

6.1.4. Comparison with Traditional Approaches

Table 9 shows a comparison between P4 and traditional anonymization defenses. Traditionally, network practitioners and researchers used offline network traffic anonymization tools (e.g., tcpanon [136] and Crypto-pan [139]). Such tools require network operators to collect and store raw packets, then perform the anonymization technique offline. Although these methods were widely adopted, they incur several burdens. For instance, wasteful usage of computation, memory, and resources, as well as the inability to anonymize traffic at line rate in high-speed networks.

On the other hand, traditional online anonymization tools, such as PktAnon [137] suffer from a low processing performance. Anonymization tools in the programmable data plane [76–79], have significant improvements in performance, yet they do not perform payload encryption and advanced privacy preserving mechanisms (e.g., anonymous web browsing) such as onion routing [138].

6.2. Cryptography and Security Protocols

6.2.1. Background

Implementing cryptographic functions in the data plane allows for achieving confidentiality and authentication at high speeds. Such functions often require complex arithmetic operations and are resource-intensive. For the switch to operate at line rate, the supported operations in the P4 language are limited (e.g., additions, subtractions, bit concatenation, etc.). Recently, a number of research contributions proposed some workaround to implement cryptographic functions and security protocols in the programmable data plane, such as Advanced Encryption Standard (AES) [140], Media Access Control security (MACsec) protocol [141], and Internet Protocol Security (IPsec) protocol [142].

6.2.2. Literature

Hauser et al. [80] propose P4-MACsec, a MACsec implementation using the P4 language. MACsec protocol [141] provides point-to-point security between MACsec-enabled endpoints within the same local area network. The same authors extend their work to propose P4-IPsec [81], an IPsec implementation using the P4 language. IPsec [142] can ensure confidentiality, integrity, and authentication for IP packets, and it is widely adopted in Virtual Private Network (VPN) protocols. IPsec comprises different protocols and modes; however, for simplification, P4-IPsec only implements Encapsulating Security Payload (ESP) in tunnel mode with two cipher suites (AES Counter Mode (AES-CTR) and NULL ciphers).

Li et al. [82] propose P4-based Network Immune Scheme (P4NIS) against eavesdropping attacks in IoT networks. P4NIS involves three lines of defenses. The first line involves multiple forwarding policies the operator can select for different IoT traffic and defense eavesdroppers. The policies could split the packets into different network paths disorderly. The second line of defense offers network operators several encryption algorithms for encrypting the transport layer of the packet's header. Finally, the third line of defense encrypts the packet's payload.

In a different line of work, Scholz et al. [83] extend three P4 targets to support multiple cryptographic hash functions and achieve authentication and resilience. The proposed work is motivated by the support of only non-cryptographic hash functions (CRC) in P4, which are often not secure and produce hash collisions. The CRC hash function is commonly used in P4 implementations since it is supported, easy to implement, and incurs low computational overhead.

The aforementioned security protocols (P4-IPsec, P4-MACsec) use a local or central controller to delegate complex operations. On the contrary, Chen [36] proposes a technique, namely scrambled lookup table, to implement AES encryption entirely in the data plane without controller interaction. AES

Table 10: Cryptographic Functions and Security Protocols Implementations in P4

Paper	Approach	Security Goal	Control-Data Interaction	Target	Limitations
P4-MACsec [80]	MACsec implementation in P4	Provide Conf., Integ., and Auth. for Ethernet frames	✓	BMv2	Does not validate a full compliance to all IEEE 802.1AE standards
P4-IPsec [81]	IPsec ESP tunnel mode implementation in P4	Provide Conf., Integ., and Auth. for IP packets	✓	BMv2, Tofino	Only supports IPsec ESP in tunnel mode
P4NIS [82]	Multiple defenses (double encryption, packets distribution)	Protect against eavesdropping in IoT	✓	BMv2	Encryption completely in the control plane
[83]	Various cryptographic hashes implementation in P4	Support protocols and applications requiring message authentication	×	t4p4s, NPU, FPGA	Key material is not used for generating message authentication code
[36]	AES encryption in P4 via scrambled lookup table	Build in-network security and privacy applications, such as IP header encryption and onion routing	×	BMv2	Side-channel attacks to retrieve keys via probing
[84]	Enhanced content permutation to protect 5G packets	Provide Conf. for 5G packet payload	×	Tofino	Limited codeword shuffling
[85]	Crypt./hash functions, digital signature called as P4 externs	Accelerate generic security implementations and facilitate their usage in P4	✓	FPGA	No line rate operation

encryption is a symmetric block cipher encryption algorithm designed to encrypt data. The proposed approach supports all three variants of the AES algorithm (AES-128, AES-192, and AES-256) that vary based on the size of the encryption key. Essentially, the adopted technique is effective in programmable switches as it reduces the number of sequential arithmetic operations needed for the AES encryption.

In 5G/IoT networks, Lin et al. [84] propose a new secret permutation mechanism in the P4 switches to protect 5G packets. The approach is similar to [36] in the sense that it is implemented in the data plane, thus, it does not incur extra packet processing overhead and operates at line rate.

In a different approach, Malina et al. accelerate cryptographic operations for FPGA-based network cards and use them as P4 externs. In particular, the authors implement main cryptographic functions, such as symmetric cipher (AES-GCM-256), digital signature (EdDSA), and a hash function (SHA3) using VHDL.

6.2.3. Cryptography and Security Protocols Schemes: Comparison, Discussions, and Limitations

Table 10 summarizes the aforementioned approaches in P4. P4-MACsec [80] features a two-tier control plane structure to implement the functionalities locally (using the switch's local control) and globally (using the centralized controller). Utilizing the local controller eliminates dependencies on external devices, reduces traffic in the management network, load on the central controller, and latency from packets exchanged with the central controller. For encryption, decryption, and packet authentication, P4-MACsec uses AES in Galois/Counter Mode (AES-GCM) [143] implemented as P4 externs.

As for P4-IPsec [81], it supports two tunnel mode operations, namely, host-to-site and site-to-site. The Security Association (SA) management is achieved via the SDN controller instead of the Internet Key Exchange (IKE) protocol in order to reduce message exchanges. Additionally, the authors restrict the support to ESP in tunnel mode and IPv4 packets only. The authors provide a working IPsec protocol on two targets. In the BMv2 software switch target, the authors used P4 externs to apply AES-CTR for encryption and decryption, and Hash-based Message Authentication Code (HMAC) using the MD5 hash function (HMAC-MD5) for packet authentication. In the Tofino target, the P4 externs of P4-IPsec are relocated to the main CPU module due to design constraints. In the NetFPGA SUME board target, the authors could not build a working prototype due to platform limitations.

In P4NIS [82], the encryption in the second line of defense is performed in the switch's controller. On the contrary, the third phase encryption is executed by the IoT device due to the limited switch resources. Evaluations show that the three lines of defenses adopted in P4NIS increase the difficulty of eavesdropping significantly compared to state-of-the-art approaches.

The cryptographic hash function implementations in [83] depends on the target, since each platform has its own way of adding P4 externs. The implemented cryptographic hash functions include SipHash-2-4 [127], Poly1305-AES [144] BLAKE2b [145], HMAC-SHA256 and HMAC-SHA512 on the CPU target t4p4s [146]. SipHash-2-4 is also supported on the NPU target [147]. Additionally, SipHash-2-4 (64-bit output) [148] and a SHA3-512 [149] are supported on the FPGA (NetFPGA SUME) target [32]. Evaluations show that although CPU targets are easily extensible, they incur the highest latency reaching up to several milliseconds. Furthermore, the NPU target offers the highest throughput, whereas the FPGA target offers the lowest latency. While cryptographic hash functions are feasible on P4 targets, further optimization techniques can be used to improve performance and resource utilization.

The implementation of AES-128, AES-192, and AES-256

algorithms requires 10, 12, and 14 rounds, respectively, to complete. As a result, the authors in [36] leverage packet recirculation, see Fig. 2, capability in P4 to pass the packet over several rounds, i.e., to simulate looping. Evaluations show that AES-128, AES-192, and AES-256 can perform at 10.92, 8.76, and 7.37 Gbps.

In [84], the controller initially generates a permutation cipher key and installs it to the switches. Accordingly, the payload of the incoming packets is partitioned into codewords and shuffled using the key at the first switch (e.g., entry switch), then the second switch (e.g., exit switch) recovers the original payload. The enhanced permutation algorithm executes in parallel and permutes the codewords across multiple stages in the ingress pipeline. Evaluations show that the switches can perform permutation at line rate without packet queueing.

Finally, the proposed approach in [85] focuses more on the VHDL implementation on FPGA-based cards and does not operate at line rate. Evaluations show that the implementation achieves 26.24 and 4.51 Gbps for cryptographic and hash functions, respectively, on a NIC with 200 Gbps throughput.

6.2.4. Comparison with Traditional Cryptographic and Security Protocols Implementations

Traditionally, cryptographic functions are implemented on general-purpose servers that are abundant in resources and can accommodate computational arithmetic operations. However, programmable switches are limited in memory and computational resources, and thus the supported hash functions in P4 are simple (non-cryptographic) due to their ease of implementation and low overhead. Currently, researchers are continuously exploring techniques and workaround to implement such functions in the data plane, as well as delegating some functionalities to the centralized controller. Furthermore, programmable Smart Network Interface Cards (SmartNICs) are gaining more attention from the P4 research community and practitioners to improve performance, resources efficiency, and security [150]. One significant value proposition of SmartNICs is CPU offload for computationally intensive tasks (e.g., hashing, cryptography, DPI etc.) [151].

6.3. Summary and Lessons Learned

Privacy and anonymity: In network anonymization schemes, existing approaches operate at high speeds to achieve anonymity via techniques such as source address rewriting and normalization, one-pad encryption, etc. An early P4-based anonymization approach (ONTAS) provides an interface for the operator to specify the anonymization policy. More recent approaches perform anonymization up to the TCP/UDP layers but require a trusted first hop or autonomous system (e.g., PANEL and SPINE). Recent ongoing research work includes DNS traffic obfuscation without degrading the throughput in the switch. The limited number of research work on anonymization techniques in P4 opens new horizons to explore and implement advanced anonymization (e.g., application layer anonymization).

Cryptography and security protocols: Early cryptographic implementations in the data plane offload complex computations either to the local CPU of the switch or to an external server, thus incurring additional latency. However, recent works have demonstrated that it is applicable to implement AES completely in the switch's ASIC; thus, achieving higher processing rates. Essentially, a common technique used in P4 to emulate loops (rounds) is to use recirculation/resubmission. Similarly, content permutation can be realized by partitioning the payload and shuffling it across multiple stages. Other approaches such as in [85] focus on accelerating cryptographic functions and calling them as P4 externs could be beneficial for organizations that can sacrifice performance in favor of security.

7. Operational Security Provisioning

The third category in the taxonomy (Fig. 4) focuses on programmable data plane techniques that envision operations to protect critical infrastructures from malicious access. This section is divided into two main subsections concerned with firewalls and defense mechanisms against generic attacks in order to authorize benign access and prohibit malicious one.

7.1. Firewalls

7.1.1. Background

A firewall allows an administrator to control access between the outside network and resources within the administered network by managing the traffic flow to and from these resources. Firewalls can be classified into stateless packet filters, stateful packet filters, and Next-Generation Fire Walls (NGFWs).

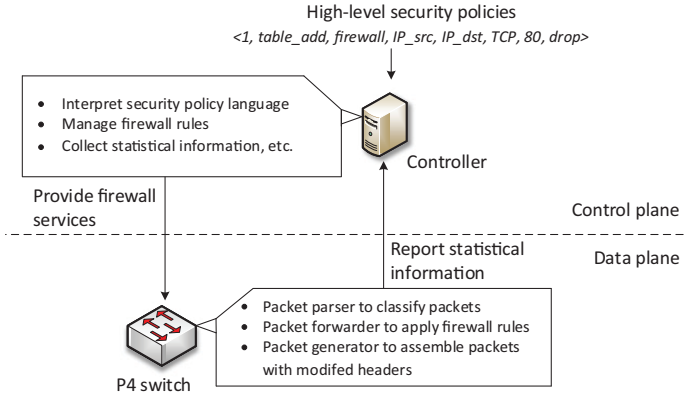

Figure 13: P4Guard architecture [86]. The controller generates and pushes match-action table rules to the switch based on the security policy defined by the operator. The switch acts as a firewall to filter network traffic and reports statistics to the controller for further analysis.

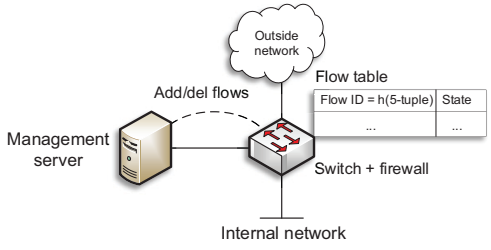

Figure 14: CoFilter Architecture [87]. The switch implements a stateful firewall by storing the state of the flows added by the management server.

A stateless packet filter examines each datagram in isolation. A stateful packet filter tracks connections. Filtering decisions on stateless and stateful packet filters are typically based on layer-3 and layer-4 header fields. NGFWs operate at a higher level. They inspect application-layer data to identify applications, users, URLs, etc. based on specific application features rather than on IP addresses or ports. The administrator configures the firewall based on the policy of the organization. Policies are then translated into rules which are stored in a table (rules table). Additional tables may be required to track the connections and corresponding statistics (e.g., flow, rule, and protocol statistics).

7.1.2. Literature Review

Datta et al. [86] propose P4Guard, a configurable firewall using the P4 language. P4Guard defines a high-level, hardware-independent, security policy language for defining the firewall rules. These rules are defined by the network operator and translated by the controller to match-action table rules that are later pushed to the P4 switch as illustrated in Fig. 13. Cao et al. [87] propose CoFilter, a stateful packet filter on the switch. For each flow, the switch stores a flow id (fid) and state information indicating the state of the connection, as depicted in Fig. 14. Similarly, Li et al. [88] propose a stateful firewall in the cloud using P4 switches. The core idea of this work is to exploit the expressiveness of the P4 language so that it parses packets and filters them based on firewall policies. In addition, the approach tracks TCP connections and allows packets belonging to initiated sessions only. The authors of [87, 88] did not specify any high-level language for defining the firewall rules.

Almaini et al. [89] propose an authentication mechanism on the switch using a port knocking service. Generally, port knocking is a mechanism performed by firewalls to authenticate hosts that send a predefined sequence of ports (TCP SYN packets) before establishing a connection. Essentially, the proposed approach stores authenticated and unauthenticated nodes (connections), represented by the IP source and destination addresses, using match-action table rules inserted by the controller. The authors later extend their approach in [90] to protect against replay attacks by implementing One-time Password (OTP) au-

Table 11: Firewall implementations using P4

Paper	Classification		Control-Data Interaction	Stateful Memory	Target	Advantages over Traditional FW	Rules Based on	Limitations
	Stateless	Stateful						
P4guard [86]		✓	Medium	Rule table, counter table (flow stats.)	BMv2	Easy to manage (policy language); more flexible firewall rules	IP, TCP, UDP, ARP header fields	Scalability issues under large packets and high transmissions
CoFilter [87]		✓	Medium	Rule table, 5 tuple hash	Tofino	Performance, latency	IP addresses and port numbers	TCP segments only
[88]		✓	Low	Rule table, 4 tuple hash	N/A	Performance	IP, TCP (src, dst)	TCP segments only
[89, 90]		✓	Low	Rule table, flow states	BMv2	Performance, latency	IP (src. and dest.), TCP port number	Manual config of conn. timeouts, adversary impersonates switch (OTP)
P4Knocking [91]		✓	Low	Rule table, flow states (IP)	N/A	Performance, latency, less complexity on hosts	IP src., TCP port number	Huge register size
		✓	Low	Rule table, flow states (IP hash)				Hash collisions
		✓	Medium	Rule table, flow states (IDs)				Controller intervention
	✓		High	Rule table				No wire speed (controller intervention)
[15]	✓		Low	Rule table	NetFPGA-SUME	Customized parser for tunnel de-encapsulation	IP, TCP, UDP, GTP, header fields	Cannot detect attacks that require heuristics (e.g., packet count)
[92]	✓		Low	Rule table	NetFPGA-SUME	Customized parser for tunnel de-encapsulation	IP, TCP, UDP, GTP, VXLAN header fields	Cannot detect attacks that require heuristics (e.g., packet count)

thentication, where a password is only valid for one transaction. In a similar extended approach, Zaballa et al. [91] present four implementations of the port knocking service that are either implemented in the data plane, control plane, or a hybrid of both.

The aforementioned firewall schemes are mainly applicable to cloud and data center networks. However, efforts in P4 are not exclusive to such infrastructures, but rather they extend to support mobile networks. Accordingly, Ricart et al. [15] propose a firewall for 5G network infrastructure located between the edge and the core networks. In a follow up work [92], the authors extended their work to support multi-tenant 5G infrastructures.

7.2. Firewall: Comparison, Discussions, and Limitations

Table 11 compares the aforementioned firewall schemes. P4-based firewalls mainly differ in terms of their control plane and data plane implementations, as well as the services they provide to the operator.

In the data plane, P4Guard [86] implements new tables for the firewall rules (based on IP, TCP, UDP, and ARP header fields), in addition to a counter table that records statistical flows (packet counters) and periodically sends them to the controller. The involvement of the controller is only required to translate the high-level firewall policies into match-action table rules on the switch, or to detect various flooding attacks (from the collected statistics).

CoFilter [87] and [88] save memory space on the data plane by computing the digest of the hash function of the 5-tuple and storing it as the fid for the corresponding flow. In addition, CoFilter uses a simple hash function available at the data plane that may produce collisions. As a result, CoFilter involves a management server, which is abundant in resources, to resolve these collisions and update the flow table and/or other tables.

As for authentication techniques, Almaini et al. [89] implement a timeout for the stored connections to accommodate the available resources and avoid attacks, such as those exhausting the memory. For enhanced security and customization, the approach allows network operators to define the length of the sequence (i.e., the number of ports that need to be knocked). As for the OTP implementation, the system uses cryptographic hashing functions (e.g., SHA3) to calculate the next expected password and is based on Leslie Lamport algorithm [152]. In [91], the first two implementations solely implement port knocking in the data plane by storing the knocking state within a register. The first implementation uses the source IP address as the index for the register. However, large register sizes (e.g., 32-bit) are impractical due to the limited memory size in existing P4 switches. As a result, the second implementation uses the digest of the CRC_{16} from the source IP address as the register index to allocate a smaller register space. The CRC_{16} produces collisions, thus, the third implementation delegates the controller to assign IDs for new clients and insert rules to a table in the data plane that matches sources IPs with their IDs. The fourth implementation offloads the port knocking service entirely to the control plane, where switches no longer keep track of the knock states.

In 5G mobile networks, GPRS is carried via the GPRS Tunneling Protocol (GTP). In the communication process, a tunnel is created between two GPRS support nodes and the original IP packets are encapsulated within a GTP header [153]. As a result, the authors of [15] and [92] customized the parser to inspect deeper into the header fields than conventional firewalls. The firewall implements a TCAM table that stores firewall rules and performs packet filtering based on several keys (e.g., VXLAN, GTP, inner and outer IP and TCP/UDP headers).

7.3. Comparison with Legacy Firewalls

Table 12 shows a comparison between P4, SDN/NFV, traditional, and next-generation firewalls. Traditional firewalls inspect network traffic up to, and including layer-4 header fields. Such a technique is ineffective against contemporary attacks that can only be detected by deeply inspecting the packet (e.g., application layer attacks). On the other hand, NGFWs are able to perform DPI and more complex operations than programmable switches. However, NGFWs are based on general-purpose CPUs that are much slower than the switch's ASIC.

Other types of firewalls include SDN/OpenFlow firewalls. Essentially, the adoption of OpenFlow by enterprise networks means that it is inevitable that legacy security appliances, such as firewalls, have to be migrated to OpenFlow-based networks. However, studies reveal that OpenFlow brings great challenges for building firewalls. For instance, the OpenFlow forwarding plane is almost stateless, rendering it incapable of actively monitoring flow status without the controller intervention [154]. Such limitations can be mitigated using the stateful

Table 12: P4, Traditional, and Next-Generation Firewalls

Feature	P4 Switch	SDN/NFV	Traditional FW	NGFW
Header inspection (up to layer-4)	Supported	Supported	Supported	Supported
DPI	Limited	N/A	N/A	Flexible
Custom header inspection	Flexible, user can define custom parser based on the network (e.g., BYOD)	Limited to OpenFlow	Fixed to the vendor	Fixed to the vendor
User-defined policies	Flexible, policies can be customized based on the parser	Limited to OpenFlow	Fixed to the vendor	Fixed to the vendor
Stateful memory	Limited to the P4 target	Limited to the switch	High	High
Speed	High (terabits)	High (without controller interaction)	Medium	Low (during DPI)

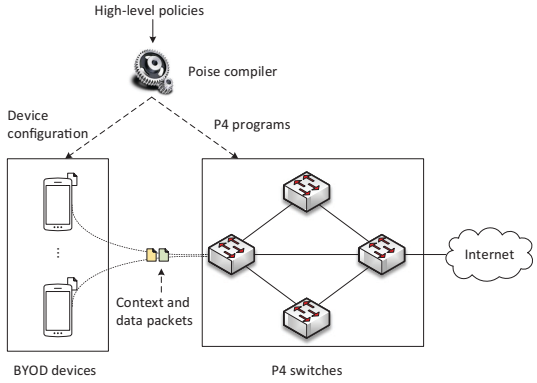

Figure 15: Poise architecture [94]. The operators define the policy they want to enforce in their network, and Poise translates the policy into a compatible configuration/program that is installed on the devices and the P4 switches.

programmable data plane, which can host additional firewall capabilities and act in real-time.

7.4. Generic Defenses

7.4.1. Background

Efforts in the P4 community have proven that programmable switches are effective against various types of network attacks. Such approaches are cost-effective (no middleboxes required), easily deployable (immediate programmability of the switch), and more efficient (line rate mitigation). In this context, a number of P4-based solutions that mitigate various network attacks are surveyed. Such attacks include Explicit Congestion Notification (ECN) protocol abuse, which can disrupt the congestion control process [96]; Optimistic ACK attack, which can mislead end-hosts to increase their TCP congestion window [97]; covert channels, which can exfiltrate secret data from compromised machines using timing and storage channels [93]; and rolling attacks, which can evade traffic engineering.

7.4.2. Literature

Xing et al. [93] propose NetWarden, a new defense mechanism that preserves the TCP performance while supporting a range of storage and timing covert channel defenses in the switch. Kang et al. [94] propose PrOgrammable In-network SEcurity (Poise), a system that defines a language for context-aware policies and translates the imposed policies to a P4 program as depicted in Fig. 15. Context-aware policy is an approach used to enforce dynamic access control on devices' runtime context (e.g., permit devices with updated operating system only). Such policies are applicable in enterprise networks, where employees use their own devices (Bring Your Own Device (BYOD)).

In contrast to the aforementioned techniques that target specific attacks (e.g., BYOD attacks, covert channels), FastFlex [95] is an approach that develops architectural support for a variety of defenses. In particular, the authors showcase FastFlex defense mechanism against rolling attacks, where suspicious traffic is routed to longer paths (topology obfuscation). Likewise, Laraba et al. [96] propose a method for modeling a stateful security monitoring function as an Extended FSM

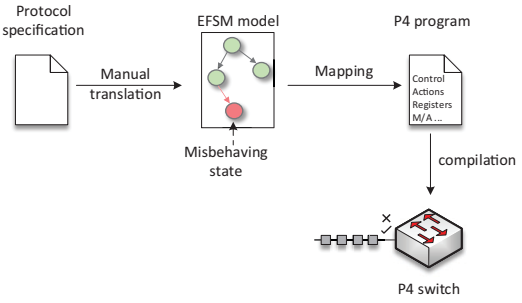

Figure 16: Modeling stateful security monitoring function as EFSM in P4 [96].

(EFSM) using P4. EFSM is simply an FSM that stores persistent values in variables rather than states to limit state explosion. Essentially, the authors require the designers to translate the protocol with its possible misbehaviors to an EFSM model. Subsequently, the proposed approach maps the EFSM to P4 as shown Fig. 16. The authors showcase their proposed approach to mitigate misbehaving end-hosts abusing the ECN protocol. The authors later extend their work in [97] to provide a formal description of the mapping between the EFSM and the P4 primitives. Additionally, the extended work can also detect and mitigate Optimistic ACK attack.

Efforts in 5G networks include FrameRTP4 [98], a P4-based system that detects and mitigates attacks in real-time within 5G network slices. The latter is a technology in which the physical network is partitioned into slices, each with its virtual resources to accommodate diverse network infrastructures [155].

A different line of research includes [99]. Essentially, security applications that resort to ML for flow classification heavily rely on collecting features such as packet length, and inter-packet arrival time frequency distribution. However, per-packet collection of these features is infeasible considering the memory constraints of current programmable switches. Furthermore, feature selection of the classifier is application-dependent. Motivated by such needs, Barradas et al. [99] present FlowLens, a system that leverages programmable switches to enable efficient flow classification for multi-purpose ML security applications. FlowLens presents a new compact representation of packet length and inter-time packet arrival distributions devised in a way that maintains the accuracy and addresses the memory constraints on the switch, referred to as flow markers. Flow markers are parametrizable based on the operator needs (e.g., operator requires a target accuracy to be attained), and automatically generated by a P4 primitive, namely Flow Marker Accumulator (FMA). The overall architecture of FlowLens is depicted in Fig. 17.

In contrast to approaches that use ML in the control plane, such as in [98, 99], Qin et al. [100] implement Binarized Neural Network (BNN) [156] in the switch at the network edge to perform line rate intrusion detection. Additionally, the approach applies federated learning [157] that allows for scalable training from multiple edge switches while preserving privacy.

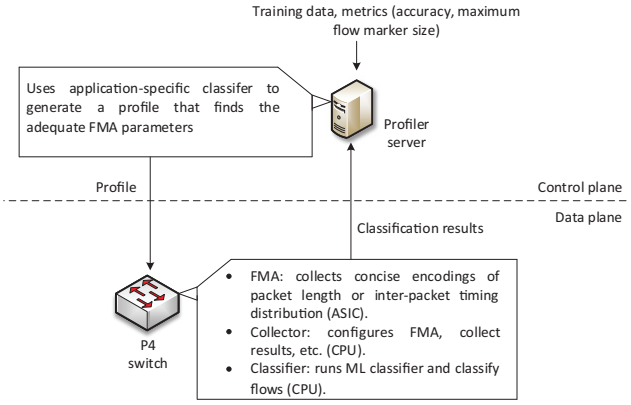

Figure 17: FlowLens Architecture [99]. The external server takes training data and other operator metrics and runs optimization techniques to generate and install a profile on the switch. The latter collects statistics based on the profile, classifies flows, and sends them to the server for further analysis.

7.4.3. Generic Defenses: Comparison, Discussions, and Limitations

Table 13 summarizes and compares the aforementioned surveyed work. NetWarden [93] performs header inspection and modification to the fields (e.g., TTL, TCP reserved) used in covert channels. For advanced storage channel overloading necessary header fields that cannot be randomized by the switch (e.g., TCP sequence number), NetWarden replaces old headers with newly generated ones and leverages the stateful memory to store a mapping between the original and the new headers for each flow. Moreover, NetWarden leverages the programmable switches to support an additional covert storage defense requiring per-packet monitoring; such a defense would not have been possible without utilizing the programmable switches. Timing channel mitigation requires statistical computations for a batch of packets (no per-packet inspection required). Accordingly, NetWarden uses the software (either switch's CPU or external server) to perform these computations. In an approach that trades the granularity for space saving, NetWarden uses CMS with a variation of the CRC hash to store connections identified by the 4-tuple (source and destination of the IPs and port).

In Poise [94], the compiler encodes context fields in customized P4 headers (e.g., Ethernet, IPv4, TCP, Global Positioning System (GPS), etc.). Context operations are translated to control blocks and P4 tables (e.g., calculating the distance from the parsed GPS header and matching it with table rules). Poise receives policy rules/updates from the controller for new connections and stores them in match-action tables as key (source IP and port, and protocol) value (allow, drop, etc.) store. Since the process of updating the match-action table requires controller intervention, Poise buffers decisions in a small cache (the key is the digest of the CRC_{16} from the 3-tuple) that can be updated at line rate until tables are updated from the con-

Table 13: Generic defense mechanisms using P4

Paper	Applications	Control-Data Interaction	Data Structure	Target	Advantages Over Traditional Defenses	Rules Based on	Limitation
NetWarden [93]	Covert channel	Medium	CMS	Tofino	Performance, accuracy	4-tuple (src / dst IPs and ports)	Communication between ASIC and CPU is a bottleneck
[94]	BYOD attacks	Low	N/A	Tofino	Customized parser (context-aware)	IP, TCP, context	Lack auth., requires external cryptography
Poise [95]	Generic, rolling attacks	Low	N/A	BMv2	Flexibility (wide range of defenses), performance, resource sharing	Based on the boosters	No cross-domain network support, requires stability across mode changes
[96]	Generic, ECN protocol abuse	Low	Hash chaining	BMv2	Seamless integration, Flexibility (stateful security modeling)	5-tuple	Manual mapping to EFSM models
[97]	Generic, ECN protocol abuse Optimistic ACK	Low	Hash chaining	BMv2	Seamless integration, Flexibility (stateful security modeling)	5-tuple	Manual mapping to EFSM models
FrameRTP4 [98]	5G NS attacks	High	BF, CMS, IBLT	BMv2	Performance (real-time), scalability, centralized orchestrater	5-tuple	Static threshold for SFCMon
FlowLens [99]	Generic, covert channel, website fingerprinting, botnets	High	FMA	Tofino	Modular, automated, scalable	5-tuple	Packets of new flows are skipped until rules are inserted
[100]	IDS	Medium	N/A	Netronome	ML (BNN) at line rate	Generic (TCP, IP, etc.)	Only handles binary output decision

troller. Poise recirculates packets a number of times to delay their processing until the table is populated from the controller (for recirculation, refer to Fig. 2). Evaluations show that Poise is agile, has reasonable overheads, and highly resilient against control plane saturation attacks (e.g., generating a large number of context changes to exhaust the controller).

FastFlex [95] transforms the defense application (P4 programs), also referred to as boosters, into Packet Processing Modules (PPMs), then to dataflow graphs and constraints (representing switch's resources). Such a decomposition can be exploited to share resources across the PPMs and generate a full dataflow graph that can be optimally mapped to the underlying network. FastFlex can dynamically scale boosters when real-time attacks are detected. Furthermore, the approach dynamically returns back to optimal default mode as soon as the attacks subside.

In [96], the states of the EFSM model (current state of the flow) and the persistent variables across the states (e.g., the TCP sequence number) are represented as P4 registers. On the other hand, variables that are not persistent across states are modeled as per-packet metadata (e.g. TTL field). Action, transitions, and conditions that trigger the transition of states are defined within P4 actions. For every monitored connection (identified by the 5-tuple), an instance of the EFSM model is maintained in the switch. Hash collisions are handled by "hash-chaining" [158], which creates a linked-list of flows (flows that collide) in the same index with different keys.

At the data plane level of FrameRTP4 [98], the P4 program implements three main functionalities. First, Service Function Chaining (SFC) [159] manages the life cycle of network slices. Such an implementation mandates a customized P4 parser for the Network Service Header (NSH) [160], required in the SFC architecture. Second, tables are utilized to implement ACLs that perform wildcard filtering (ternary match) based on the 5-tuple header. Third, network flows are stored and monitored (SFCMon module) in the data plane. For efficient and scalable monitoring, BF, CMS, and Invertible Bloom Lookup Table (IBLT) data structures are leveraged. The control plane manages the lifecycle of the data plane functionalities (e.g., insert/delete rules). Moreover, the controller implements an ML classifier to detect new attacks offline and update the switch's rule table.

FlowLens [99] (see Fig. 17) utilizes match-action tables and a grid of P4 registers to perform two operations, namely quantization and truncation. The former counts the packet length in coarse bins (P4 registers), whereas the latter selects the bins considered as the most relevant features for the ML classifier. The FMA generates flow markers for each flow based on quantization and truncation parameters. Essentially, for each incoming packet, the FMA hashes the 5-tuple to obtain the fid, matches it in the match-action flow table, and returns the flow offset (representing the row of the register grid). The bin offset (representing the row of the register grid) is computed through aggregation, bit shifting, and table matching operations (using the control plane of the switch). FlowLens was tested on three security applications, covert channel detections, website fingerprinting, and botnet detection. The compression scheme allows FlowLens more flows than the baseline (no compression, raw packet distribution) while attaining high accuracy in predicting the classes of traffic flows.

The approach in [100] applies BNN since it can be efficiently implemented in the switch. Essentially, BNN has a binary decision, and the computations are converted into bitwise operations supported by the switch. Since training the BNN requires complex operations, it is handled by the control plane, and the weights are pushed to the data plane (saved in registers).

7.4.4. Comparison with Legacy Approaches

Defense mechanisms against generic attacks within legacy networks can be implemented on the end-host, the router/switch, or using a middlebox. Typically, host-based defenses and middleboxes add extra probing traffic into the network and are not as fine-grained as in-network approaches.

In SDN/OpenFlow network monitoring, the majority of router-based defense solutions [161, 162] are stateless. Thus, they are incapable of tracking complex protocol behaviors unless third-party middleboxes are added.

In contrast to in-network middleboxes, the security boosters (defenses) such as in FastFlex are not fixed in functionality or location. Additionally, FastFlex orchestrates network-wide defenses in the data plane rather than the control plane for higher throughput.

Covert channel defenses require per-packet inspection and a customized parser to detect exfiltrated data. However, general-purpose CPU-based defenses mostly work offline over low speeds or small samples of network traffic. On the contrary, legacy routers/switches can handle high-speed traffic, yet cannot perform customized parsing.

7.5. Summary and Lessons Learned

Firewalls: In efforts to reduce the cost incurred by hardware firewalls, as well as to expedite the performance while allowing legitimate traffic and denying malicious ones, several works have implemented firewall functionalities in P4. Such functionalities include packet filtering, port knocking, and defenses used by 5G network firewalls. Furthermore, programmability in the data plane allowed vendors to produce custom-built firmware based on the P4 description delivered by the customer. This opened new horizons for network engineers to design firewalls with rules based on header fields specified by the customer, i.e., non-traditional fields.

Generic attacks: In the context of operational security provisioning, the programmable data plane was proven to be effective against various types of attacks with little to no performance penalty in the network. Such attacks target the application layer, BYOD enterprises, 5G networks, Internet eXchange Points (IXPs), ISPs, etc.

In the meantime, P4-based approaches, such as those acting as firewalls, might not completely replace their state-of-the-art NGFW or middleboxes due to current memory and computational limitations. However, they can act as an additional line of defense in highly targeted infrastructures to speed up the protection and mitigation process.

Table 14: P4 Applications and security implications under the STRIDE model

P4 application	Potential Security Implication	Threat	Remediation
Network telemetry	Unencrypted metadata interception through a man-in-the-middle	(1) Information disclosure (2) DDoS	PBT [163]
Load balancing	Exploit the used hash in signature matching, man-in-the-middle to drop probing packets	DDoS	(1) Authentication (2) Cryptographic hashes (3) Automated attack discovery [164]
Congestion control	Telemetry, threshold, and throughput rate tampering	(1) Tampering (2) DoS	(1) VHE [165] (2) Authentication
In-network cache	Cache tampering, absense of cache write logs, repeated cache and repudiation	(1) Tampering (2) Repudiation (3) DDoS	(1) Authentication (2) Controller delegation (3) DNSSE [166]
Cryptography	CRC function exploitation, infinite loop creation (e.g., when emulating multiple rounds via recirculation)	(1) Tampering (2) DDoS	(1) Pseudo-cryptographic hash (2) Network verification
Telecommunication	Turning the switch into a powerful relay for malicious purposes	DDoS	Authentication
Consensus	Compromising the switch and launch Sybil attack	(1) DDoS (2) Tampering	(1) Authentication (2) Proof-of-work
Application-agnostic	Buggy P4 programs (e.g., reading invalid headers)	(1) Spoofing (2) Elevation of privilege	(1) Network verification (2) Use production-ready targets

8. Stride-based Model Security Analysis of P4 Applications

STRIDE is currently the most mature threat modeling process [167]. Such a model identifies spoofing, tampering, repudiation, information disclosure, DoS, and elevation of privilege attacks. Previous work [22] employs the STRIDE model on P4 components only (e.g., parser, compiler). On the other hand, this work performs threat analysis on P4 applications that are gaining wide interest from the industry, researchers, and network practitioners. Such an analysis is needed as it highlights the major threats in each P4 application. Beyond this analysis, a number of mitigation techniques is presented for each threat in each application. Table 14 summarizes the P4 applications, their corresponding security implications, and the remediation strategy according to the STRIDE model.

8.1. Network Telemetry

8.1.1. Background

Network telemetry has proliferated to fulfill the ever-increasing demands of network measurements, such as better scalability, accuracy, coverage, and performance. It is an automated process in which the measurements are performed at remote or inaccessible points and transmitted to receiving equipment for monitoring. In-band network telemetry is an emerging representative of network telemetry that deduces the network status by inserting metadata into the data packet through switch nodes in the path. Notable research contributions include In-band Network Telemetry (INT) [168] led by P4.org, In-situ Operation Administration and Maintenance (IOAM) [169] led by the IEFT, Alternate Marking-Performance Measurement (AM-PM) [170] and Active Network Telemetry (ANT) [171].

8.1.2. Vulnerability Assessment

INT scheme is illustrated in Fig. 18. The INT architecture consists of INT source, INT sink, and INT transit hop. Essentially, the INT source embeds telemetry instructions into the packets, whereas the INT sink extracts and reports the telemetry results. In the intermediate state, the INT transit appends telemetry metadata according to the instructions inserted by the INT source.

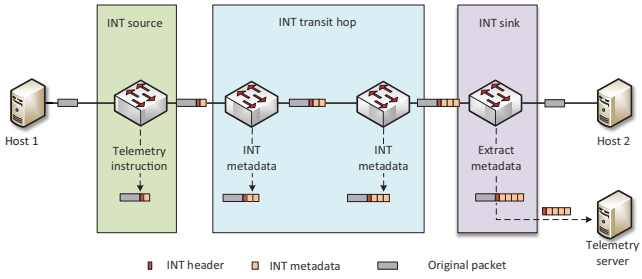

Figure 18: In INT, the source switch marks the packet for telemetry collection. Then, each transit switch piggybacks the metadata on the packet's header. The INT destination removes the INT header and sends it to a telemetry server, as well as sends the original packet to the destination.

[165] and [163] reported that INT suffers from a number of security vulnerabilities. INT gets reported to the control plane through control channels based on the Transport Layer Security (TLS) or Secure Socket Layer (SSL) connections, which are vulnerable to man-in-the-middle attack. If an attacker exploits this vulnerability, they can disclose information and launch severe attacks, such as DoS attacks. Furthermore, the attacker could be more aggressive to modify the telemetry data for misleading the data analytics system; thus, severely disturbing the network automation process.

8.1.3. Plausible Remediation Approaches

Postcard-Based Telemetry (PBT) [163] is an improvement for IOAM technology, in which the data plane processing overhead and user data packet are reduced. PBT scheme is illustrated in Fig. 19. Essentially, PBT collects telemetry data

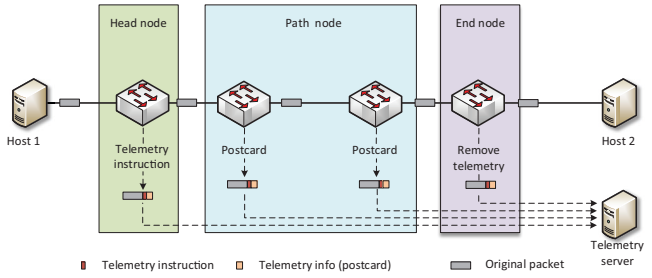

Figure 19: In PBT, the head node inserts the telemetry instruction. Each switch along the path generates telemetry information (postcard) and sends them to the telemetry server separately. The destination switch removes the PBT header and sends the original packet to the destination.

by either marking user packets (PBT-M) or inserting telemetry instructions into user packets (PBT-I) rather than appending the actual telemetry metadata to the packet's header. For every marked or instructed packet, the switch generates telemetry data and exports them directly.

The authors of PBT claimed that the present security vulnerabilities in INT are mitigated in their approach. As the telemetry data is exported separately from the user packets, the process of encrypting and securing the collected data without being exposed to external entities is easier. Thus, deterring passive eavesdropping, and avoiding critical attacks [163].

8.1.4. Summary and Lessons Learned

Existing network telemetry approaches have a wide range of applications, such as DDoS, congestion control, traffic engineering, etc. The amount of telemetry inserted in the network is application-specific. Malicious nodes can exploit this behavior by constructing telemetry packets and overwhelming the network, possibly creating flooding attacks. Existing telemetry approaches such as PBT can offer an increased layer of security using encryption. Nonetheless, it is not enforced on the operator. Future work around this area could explore a standard that preserves the authenticity of the participating telemetry nodes so that operators can protect their networks against telemetry attacks regardless of the technology used.

8.2. Load Balancing

8.2.1. Background

Data centers often support requests from numerous users by directing traffic to dedicated servers (load balancers) to distribute the requests across multiple servers; thus, balancing the load among them. Recently, programmable switches were proven to implement load balancing functionalities by leveraging stateful processing [14, 172, 173]. Essentially, the switches store the state information directly in the data plane (e.g., representing the best path to a destination) instead of dedicating an external device.

8.2.2. Vulnerability Assessment

HULA [172] is a load balancing approach that works similarly to distance-vector routing. It uses periodic probing

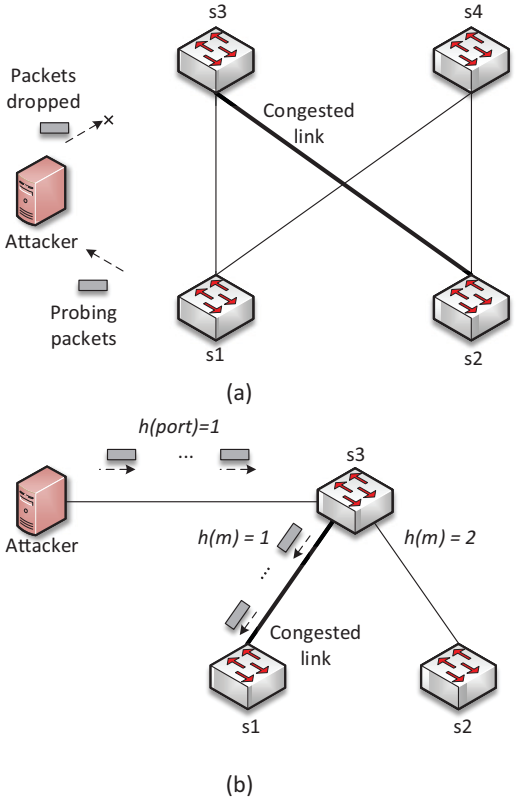

Figure 20: Attacking the load balancing application in the data plane by two methods. (a) Disrupting the communication via a man-in-the-middle attack. (b) Exploiting the simple hash function used in load balancing.

to proactively update the network switches with path performance information and optimize their routing table on the fly. SilkRoad [14] is a significant ASIC-based load balancer that maintains per connection consistency during frequent direct IP address updates by leveraging the stateful processing in P4. BLINK [174] detects TCP-retransmission and uses it as a link failure indicator. Indeed, such techniques have a plethora of benefits in terms of speed and cost for data centers and enterprise networks; however, they lack data signals authentication. Thus, facilitating network attacks without requiring the attacker to have high privileges in the network.

A man-in-the-middle could craft packets that contain falsified information about the available links or even drop probing packets between the switches. In such a scenario, legitimate switches will overload the network with update packets to establish new paths. Furthermore, the attacker could reroute the traffic through a certain path/switch, thus, creating a DDoS attack. Such an attack is depicted in Fig. 20(a), where an attacker (man-in-the-middle) drops all the probing packets sent from switch s1 to switch s3; thus, mimicking link failure between the two switches. As a result, switch s3 redirects all the traffic towards switch s2, possibly creating a DoS attack.

Furthermore, simple hash-based load balancers that do not involve cryptographic functions can be exploited to launch DDoS attacks. Such a technique hashes the packet header and picks one of the available paths based on the hash value. An attacker aware of the used hash function can craft skewed traffic

patterns where the hash of the header fields is constant across the trace [164]. In Fig. 20(b), switch s3 load balances the traffic between switches s1 and s2 using a simple modulo function, for instance, `port%2`, where `port` is the source port of the packet. An attacker can craft packets in which the `port` always hashes to 1; thus, exploiting the used hash function and congesting the link `s2-s3`.

8.2.3. Plausible Remediation Approaches

To remediate the security breaches on switch-based load balancers, network operators can use network authentication of the processed signals. This ensures that an attacker will not be able to falsify information between the switches as in Fig. 20(a).

To mitigate attacks against load balancers utilizing simple hash functions, network engineers can utilize cryptographic hash functions (e.g., those developed in [83]) that are more secure and hard to break. Therefore, an attacker cannot easily craft packets that always hash to the same value as in Fig. 20(b).

Alternatively, Kang et al. [164] develop an approach to mitigate the vulnerabilities in load balancers implemented in the data plane without the need to authenticate the communication, as it would downgrade the performance of the switches. The approach comprises three main steps. The first step derives the expected behavior of the data plane system via probabilistic symbolic execution. The second step is to negate the expected behavior (e.g., in HULA, the expected behavior is modeled by repeatedly hashing packets to the same value or pattern). The third step is to synthesize runtime monitors to detect in real-time malicious traffic.

8.2.4. Summary and Lessons Learned

P4-based load balancers have gained a lot of attention in recent years due to high-performant ASICs that can replace thousands of software-based load balancers [14]. Despite the increasing emergence of switch-based load balancers, few papers discuss the security aspects of load balancers (e.g., use of strong encryption). The proposed approaches implement load balancing in programmable switches and do not take into consideration hybrid networks. This could restrict the deployment of the approaches as it exacerbates the cost, as well as introduce potential security issues (e.g., handling compromised legacy and programmable switches).

8.3. Congestion Control

8.3.1. Background

Network congestion is an ongoing problem that requires applicable solutions, especially with the increase of applications that require high Quality of Service (QoS). A network suffers from congestion when a node handles traffic beyond its capabilities; thus, causing queue delays and packet loss. Programmability in the data plane offers flexibility to offload and design congestion control solutions on the programmable data plane switches.

8.3.2. Vulnerability Assessment

High Precision Congestion Control (HPCC) is presented in [175] to achieve low latency, high bandwidth, and network stability in high-speed networks by utilizing flowing packets to gather relevant information. During the propagation of the packet from the sender to the receiver, each switch along the path leverages INT to report the current status of the egress port. Metadata can include timestamp, queue length, transmitted bytes, and the link bandwidth capacity. Upon the packet's arrival at its destination, the receiver copies the metadata information to an ACK message and sends it back to the sender. When the initial sender receives the ACK message, its flow rate is adjusted based on the gathered network load information. Fig. 21 showcases two potential shortfalls that could emerge from the design of any unsecured congestion control mechanism on the data plane. Firstly, the lack of encryption could lead to telemetry data tampering and ultimately mislead the sender to disrupt the congestion control mechanism. Secondly, the absence of authentication at the receiver side, which is crucial in identifying the switches that participate in the telemetry data collection.

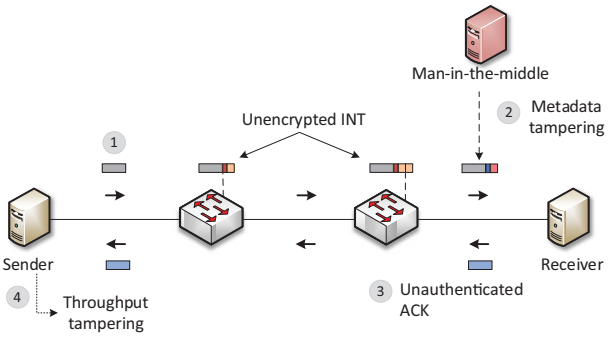

Figure 21: Congestion control throughput tampering due to unencrypted meta-data and lack of authentication.

Similarly, the authors in [176] propose a solution that relies on real-time access to gather important packet meta-data (e.g., queue load) and detect congestion only when the delay surpasses a certain threshold. That said, instant traffic rerouting is performed in the data plane to address the congestion problem. Fig. 22 accentuates the problem of threshold-based function which could lead to unnecessary traffic rerouting and traffic overhead on the central controller.

Kfoury et al. [177] use programmable switches to enable TCP pacing, a technique used to minimize traffic burstiness and packet losses. Since the authors use a custom protocol to add the number of large flows through unencrypted messages, a man-in-the-middle can tamper the packets to increase/decrease the number of senders and change the behavior of the network. For instance, altering the number of senders from 10 (real) to 100 (fake) will reduce the capacity of the link by an order of 10. Similarly, the same scenario can be used to decrease the number of senders from 100 (real) to 10 (fake) to increase the transmission rate by the senders, and thus, create a DDoS attack.

8.3.3. Plausible Remediation Approaches

In order to circumvent the threat of tampering in threshold-based functions employed in congestion control solution in the

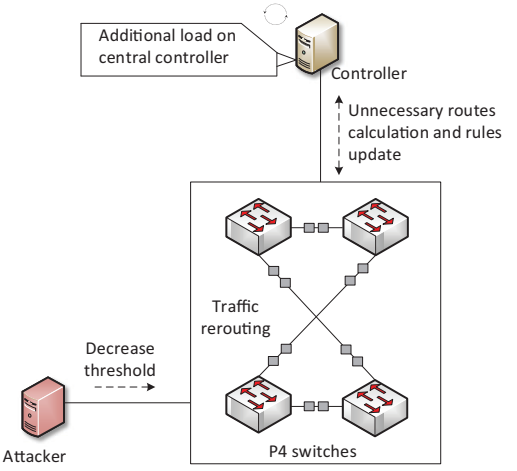

Figure 22: Adding network traffic load through threshold manipulation in congestion control systems.

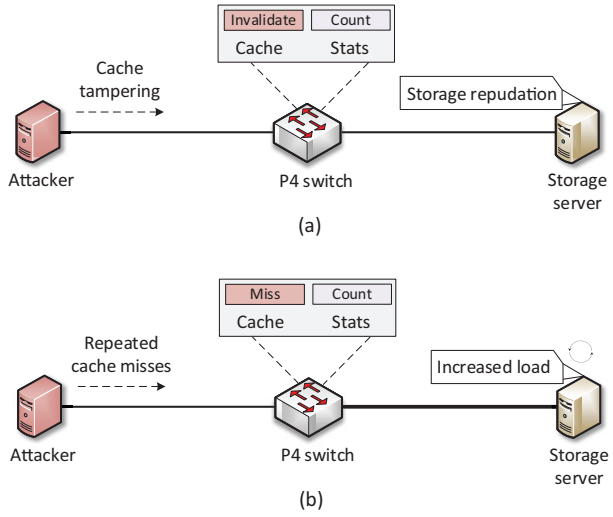

Figure 23: (a) Cache tampering and storage repudiation attacks. (b) DDoS attack due to increased load on the storage server.

data plane, lightweight encryption schemes should be enforced such as the Vector Homomorphic Encryption (VHE) [165]. Moreover, available security practices should be followed to enable authentication on end hosts.

8.3.4. Summary and Lessons Learned

With the increasing diversity of network traffic, the proliferation of IoT devices [178], and high-speed links, limiting the congestion in the network becomes a challenging task. Existing congestion control approaches relying on strategies such as INT [175] share similar security implications as those stated in the network telemetry section. Consequently, likewise security approaches can be followed. For instance, authenticating the nodes participating in the congestion control mechanism (end-hosts and programmable switches) and enabling lightweight cryptographic algorithms to avoid computation overhead.

8.4. In-network Cache

8.4.1. Background

In-network caching is a technique used for storing regularly requested information resulting from previous queries or computations for later use to improve access time to this information. Also, caching can help decrease link traffic on servers, thus, enhancing the QoS for the end-user.

8.4.2. Vulnerability Assessment

NetCache [16] presents an application-level functionality for in-network caching. It mainly relies on programmable switches to deliver on-path key-value items by leveraging the match-action tables to classify keys carried in packet headers. That said, the register arrays implemented as on-chip memory in programmable switches are used to store the values. In addition, NetCache employs a controller to update the caches with hot items (i.e., frequently accessed items) and storage servers to guarantee cache consistency and the mapping of query packets to API calls for the key-value store. In another work, DistCache [179] is a load balancing scheme for large scale storage systems using cache allocation and query routing. More precisely, DistCache achieves cache allocation by partitioning hot objects with independent hash functions between cache nodes, and adaptively route queries with the power-of-two-choices. The cache switch mainly implements a key-value module along with other modules and communicates with the controller using the thrift API generated by the P4 compiler to compute and notify on cache partitions.

In-network cache solutions still suffer from shortfalls in their designs and implementations and require security measures to reduce attack risks. As such, the lack of prior authentication in cache nodes depicted in Fig. 23(a) might result in cache tampering and repudiation threat. Specifically, when a write query initiated by an incoming packet invalidate stored copies in the switches that are on the route to the storage servers. Additionally, Fig. 23(b) depicts the threat case of repeated cache misses which can increase the load on the storage node and potentially result in a point of failure if not equipped to handle increased load. Moreover, in DNS caching implementations, the absence of adequate security aspects may lead to cache poisoning attacks by sending a large number of resolution requests with spoofed source IP addresses to resolve a specific name.

8.4.3. Plausible Remediation Approaches

Authentication measures in switch-based cache nodes can limit unauthorized write queries and ultimately prevent cache tampering and repudiation threats. A potential case for handling single point of failure would be presented by a mitigation action initiated by a central controller where the partitions of a failed switch or storage node are redistributed to an alive surrogate. The adoption of pre-existing security remedies such as DNS Security Extensions (DNSSEC) [166], which provides data authentication and digital signature, is of high importance in order to preserve the integrity of switch-based DNS caching systems.

8.4.4. Summary and Lessons Learned

The stateful memory in programmable data planes has been utilized to act as an in-network cache for storing frequently requested items. This enabled the processing of billions of queries within seconds without overwhelming the resources [16]. Data tampering and DDoS attacks can be launched on in-network caching switches lacking authentication. Such attacks are often out of the scope of the papers (e.g., in [180]), however, they can wreak havoc, especially with the increasing switch performance and limited memory resources. Future endeavors could explore storing the hot items in a distributed fashion among the switches in the network for better performance under limited memory resources.

8.5. Cryptography

8.5.1. Background

Data plane implementations of cryptographic functions are useful for various purposes, such as achieving confidentiality and authentication. Such functions often require complex arithmetic operations and are resource-intensive. In order to operate at line rate, the supported operations in P4 are limited (e.g., additions, subtractions, bit concatenation, etc.). Thus, the hash functions in P4 are non-cryptographic (e.g., CRC). Recently, a number of papers proposed workaround techniques to implement cryptographic functions and security protocols in the programmable data plane.

8.5.2. Vulnerability Assessment

The advent of programmability in the data plane has broadened the scope of applications implemented programmable switches to include heavy-hitter detection, in-network caching, in-band network telemetry, etc. A significant number of these applications, such as those in [50, 52, 126, 181], mainly use hash-based data structures (e.g., hash tables, BF, or CMS, etc.) to track flows [83]. The majority of these works either do not detail the used hash function in their implementation or use the CRC hash function implemented in P4 [83]. CRC is a non-cryptographic hash function with a secret modulus k modeled by:

$$f_k(d) = d \mod k$$

In hash-based sampling, each packet d is sampled if the hash of the packet d falls within a selection range based on the formula:

$$Samp(d) = \begin{cases} 1, & f(d) \in [R_1, R_2]. \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

where f in the case of CRC is a keyed hash function taking on values in $\{1, ..., 2^n\}$. An attacker can exploit the linearity of the CRC hash function in two different ways based on the attacker's knowledge [182]. First, if the attacker does not have any history of the victim (the hash key k is unknown), the attacker starts by sending a packet p, followed by a linear progression of packets $p + 1, ..., p + sr * 2^n$, where sr is the sampling rate. The attacker selects a range $[R_1^*, R_2^*]$ and sends packets that belong to this range. Using this process, the probability that the range $[R_1^*, R_2^*]$ will be different from the true range $[R_1, R_2]$ used by the sampler is $1 - 2 * sr$. The attacker needs to try at most $(log(0.01)/log(1 - sr))$ packets before sending a packet p that falls in the range $[R_1^*, R_2^*]$, and even fewer than this value in case the key k is known. Second, findings in [182] show that the attacker can learn the selection range and the CRC hash key using past history and some leaked information from the sampler. Specifically, the attacker can estimate the value of the key and each selection range within:

$$1 + (m + 1) + log_2\left(\frac{1}{sr}\right)$$

where m is a value defined to set a lower bound on the size of the hash key ($k > 2^{n-m}$). In further analysis, the authors state that the keys and ranges can be exactly learned within:

$$6 + 2 * (n + m)$$

Exploiting the aforementioned vulnerabilities by an attacker can lead to several evasion scenarios which could be listed under data tampering or DoS. For instance, (1) generating packets streams that evade selection by the sampler in order to avoid being billed by providers for network utilization; or (2) evading the sampler used by an IDS to perform a DoS attack.

The authors in [36] proposed a technique, namely Scrambled Lookup Table, to implement AES encryption in P4 network switches. AES-128, AES-192, and AES-256 algorithms require 10, 12, and 14 rounds, respectively, to complete. The authors leveraged packet recirculation capability in P4 to pass the packet over several rounds, i.e., to simulate looping. Loops in P4 are powerful features necessary to a variety of applications, such as implementing Multiprotocol Label Switching (MPLS), and multiple levels of encapsulation/decapsulation. However, creating infinite loops is possible on some targets, which could disrupt the functionality of the P4 switch and cause a DoS attack as depicted in Fig. 24. In particular, three techniques can be used to create loops in P4, namely, packet resubmission, packet recirculation, and packet cloning. As a proof of concept, Dumitru et al. [23] report that they coded a program that infinitely loops packets on a software switch (BMv2), as well as a hardware switch (Tofino).

8.5.3. Plausible Remediation Approaches

It is often recommended to use cryptographic hash functions instead of non-cryptographic ones (e.g., CRC) as they aim to guarantee a number of security properties, such as collision resistance. However, cryptographic hash functions are slow (affect line rate processing), and complex (consume the target's resources). Pseudo-cryptographic hash functions, such as SipHash, are optimized for small inputs, as well as for performance in software. [83]. Alternatively, network operators may use more complex hash functions, such as HMAC-SHA512, to level up the security by sacrificing the line rate processing of the switch.

To close loops created by buggy P4 programs, network engineers can use network verification techniques. For instance,

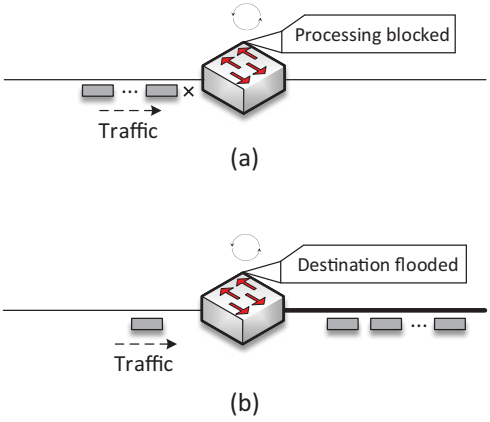

Figure 24: DoS attacks can occur due to loops in the P4 program in two ways. (a) Incoming packets are dropped due to packets looping the ingress pipeline; thus blocking the processing. (b) The destination is flooded with packets due to packets being cloned on the egress pipeline.

Vera [69] can always detect loops on the main input port, as long as there is enough memory.

8.5.4. Summary and Lessons Learned

Cryptographic implementations in the data plane have allowed network operators to achieve various security objectives without a need for end-host interaction. Due to the limited resources in the data plane, implementing heavy cryptographic functions in the data plane might drastically affect the performance of the switch. For instance, implementing loops in P4 using recirculation not only degrades the throughput but also can be exploited to launch DDoS attacks. As a result, future endeavors could explore feasible lightweight cryptographic function implementations, while studying the performance penalty.

8.6. Telecommunication Services

8.6.1. Background

Telecommunication providers always strive to improve the network performance and provide new services by updating their network infrastructure. To increase cost-efficiency and flexibility, NFV implements network functions on commodity processors (based on x86 or ARM architectures) as they have low cost and can establish innovations at a faster pace compared to purpose-built, fixed-function hardware products. However, implementing NFV on general-purpose CPUs does not meet the performance requirements of high-throughput data plane components in large carrier access networks. As the highly flexible P4 programmable data planes meet the challenging demands imposed in telecommunication environments, several proposed approaches are offloading NFV components from commodity processors to programmable switches [183].

8.6.2. Vulnerability Assessment

Shah et al. [184] redesign the Long Term Evolution Evolved Packet Core (LTE EPC) mobile packet core to improve the performance of the control plane by offloading a number of its procedures to the data plane. Singh et al. [185] implement a virtual Evolved Packet Gateway (vEPG) in P4. SMARTHO [186] is a scheme that improves handover (i.e., the process of transferring data from one cell to another while the mobile user is moving). In another line of work, Kfoury et al. [187] offload media traffic (e.g., voice over IP) from relay servers to programmable switches.

Indeed, the deployment of 5G networks and the utilization of programmable switches significantly improve the speed, capacity, and latency. However, this also opens for more severe volumetric attacks. For instance, an attacker can turn a switch into a very powerful relay of DDoS traffic in the absence of switch authentication.

8.6.3. Plausible Remediation Approaches

Telecommunication providers should take into consideration the security implications of offloading services to the data plane. For instance, the potential DoS attack in [184] can be remediated by tracking the number of unoffloadable handover messages corresponding to a certain flow. Once the tracked number exceeds a certain threshold, the operator can be notified to act accordingly. Furthermore, authentication must be employed to protect switches from being compromised such as in [187].

8.6.4. Summary and Lessons Learned

The recent trends and research efforts have demonstrated that programmable switches are effective in implementing a number of telecommunication functionalities, especially those pertaining to 5G, due to their low latency and high processing speeds. Without taking the necessary security measures, programmable devices can be turned into an attack point to severely disrupt the network. The authors in [188] pinpointed that their design of multi-tenant 5G architecture using FPGAs imposed serious TCAM memory challenges. Consequently, future work can explore efficient algorithms to prevent possible saturation attacks on the switches.

8.7. Consensus

8.7.1. Background

Distributed systems have a fundamental problem of achieving reliability (i.e., to deliver the intended service) even in the presence of faulty processes. To solve this problem, coordinating processes need to reach consensus to be able to reliably agree on some value [189]. Consensus protocols are inevitable to build fault-tolerant, distributed applications and services such as Google's Chubby [190], OpenReplica [191], Ceph [192], etc. A downside of consensus protocols that has ever lasted is the additional latency they impose on the system since they require coordination on every request. Recently, several research papers have explored leveraging programmable switches to achieve high-performance consensus [193–197].

8.7.2. Vulnerability Assessment

Dang et al. [193] present an implementation of Paxos using the P4 language. The proposed solution defines a custom header for Paxos messages and encapsulates them inside a UDP packet. Furthermore, P4 registers are utilized to store the history of values used in Paxos to achieve consensus. The authors

build a partial implementation of Paxos on the switch (phase 2 Paxos). A more recent work that includes the same group of researchers proposes P4xos [194]. The proposed work provides a complete Paxos implementation for the data plane (phase 1 and 2 Paxos) without strengthening assumptions about the behavior of the network. Li et al. [195] propose Network-Ordered Paxos (NOPaxos), an algorithm that relies on network ordering to achieve strong consistent replication without the need for coordination. NetChain [196] provides coordination in datacenters while minimizing the latency. Eris [197] can process a large class of distributed transactions while avoiding both replication and transaction coordination overhead.

Consensus algorithms are often designed according to certain assumptions in the network. For instance, Paxos assumes that given $2f+1$ processes, at most f faulty processes can exist. The aforementioned approaches do not discuss the security implications of distributed systems. Consequently, compromised P4 switches form a threat to the assumptions made in the consensus protocols. An attacker controlling the majority of the switches can out-vote the honest nodes and disrupt the consensus process. Thus, the attacker can choose not to agree on a value, making the consensus protocol run indefinitely, thus, and potentially causing a DoS. Another attack on distributed systems is the Sybil attack [198], in which an identity (e.g., adversarial P4 switch) is forged to form multiple identities in the system. Using the forged identities, Sybil attack can also disrupt the consensus process and lunch several attacks affecting the availability of the network (out-voting honest nodes). For instance, in distributed systems such as blockchain, Sybil attack can revert transactions, which can lead to double spending (tampering of data).

8.7.3. Plausible Remediation Approaches

One approach that prevents faulty processes from impersonating honest replicas is P4 Byzantine Fault Tolerance (P4BFT), in which the controller authenticates each message using the message authentication code. While this approach inevitably increases the latency and communication overhead due to the controller interaction, it definitely secures in-network consensus protocols. Other proposals to defeat Sybil attacks include proof-of-work functions, in which every process proves to others that a certain amount of computational efforts (e.g., resources) has been expended for some purposes. The proof-of-work functions require complex computations and are usually implemented on x86 processors. Thus, although they might have a feasible implementation on the switches, they can significantly degrade the throughput.

8.7.4. Summary and Lessons Learned

Consensus has recently gained substantial treatment from the P4 research community. Consensus in the data plane was proven to significantly improve the performance of the core infrastructure and services, reduce bottleneck for distributed applications, amplify the throughput, etc. [193–195]. Without authenticating the participating node, the consensus protocol can fail and possibly create a DDoS attack.

8.8. Application-agnostic

8.8.1. Background

Traditionally, closed-source fixed-function data planes are tested and verified over a long period of time before deployment. Additionally, having a fixed-function data plane reduces the likelihood of bugs produced by network operators. On the other hand, the flexibility introduced with data plane programming comes at the cost of the robustness of the P4 program. Buggy P4 programs can be exploited to breach various security measures and hence perform severe attacks [23].

8.8.2. Vulnerability Assessment

Reading arbitrary header values in P4 or resurrecting dropped packets (architecture-specific) can help attackers perform spoofing attacks through bypassing ACLs enforced by the P4 program. For instance, the attacker can revive certain packets dropped by the controller (e.g., marked as malicious) to continue their pipeline processing and eventually execute the intended attack [23].

Likewise, the same factors that allow the attackers to perform spoofing can permit them to elevate their privileges. For instance, crafting packets to bypass the security filters employed in the network.

8.8.3. Plausible Remediation Approaches

Attacking P4 programs depends on two factors, the code written and the target it is deployed on. P4 network developers must use network verification techniques to infer potential bugs and patch all possible security vulnerabilities. Furthermore, for robust secure deployments, it is preferred to depend on production-ready targets (e.g., Tofino target).

8.8.4. Summary and Lessons Learned

The flexibility, programmability, and performance gain offered by programmable data planes have made it one of the most sought-after fields in networking. As discussed earlier, applications implemented in P4 include those pertaining to cybersecurity (e.g., DDoS detection), as well as other applications (e.g., INT, load balancing, cryptography, etc.). Regardless of the P4 application, the code is susceptible to bugs that are undesirable for security purposes. It is highly encouraged that network operators use network verification techniques to unveil bugs. Future work can explore more verification techniques and expressive languages tailored to network operators, so that they can validate the behavior of the program prior to deployment, as well as during runtime using a high-level ACL language.

9. Challenges, Current Initiatives and Future Work

In this section, we present the challenges pertaining to programmable switches and their effects on security implementations in P4. Furthermore, a number of current initiatives and future directions addressing the discussed challenges are discussed.

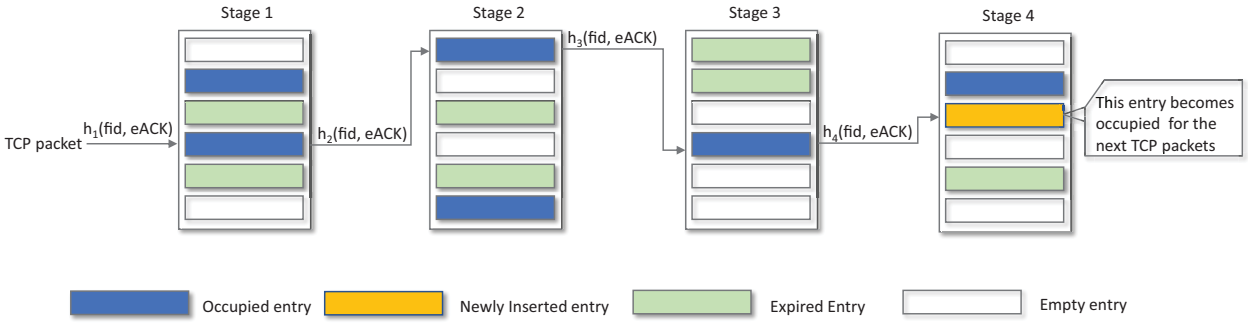

Figure 25: In multi-stage hash table, several hashes are utilized until an available entry in the memory is located.

9.1. Memory Size and Accessibility

The limited on-chip memory in the switch (e.g., limited number of SRAM per stage, and limited number of stages), as well as the restrictions on the memory access (e.g., a packet can only access few addresses in the memory), have affected several applications. For example, several straight-forward functions cannot be performed (e.g., finding the minimum across all elements). Among the affected applications are those pertaining to network security implementations. For instance, it is not feasible to maintain a significant per-flow state, i.e., to track and store every flow in the switch.

In DDoS attack mitigation techniques, the switch's memory is utilized to store information about IP prefixes and their corresponding counters, e.g., how frequently they are requesting the network. Memory constraints limit the number of monitored IP prefixes. Similarly, in traffic anonymization, switches rewrite certain packet headers and store the mapping between the original headers and the rewritten ones in the forwarding table. Considering the memory constraints in the switch, it is challenging to anonymize a large number of network prefixes as their mappings need to be stored in the forwarding table. Finally, in data protection, notable research papers have implemented encryption techniques using the stateful data plane. For instance, P4-IPsec [81] stores packet counters for the SAs in the data plane using registers. The SA is fundamental to IPsec as it describes how the entities will use security services to communicate securely. Accordingly, encrypted communication is limited to the number of SAs that can be stored in the switch.

9.1.1. Current Initiatives and Future Work

To utilize the memory and resolve the packets that hash to the same memory location, Chen et al. [199] implement a multi-stage hash table using multiple memory arrays across different pipeline stages to store the records of outgoing TCP packets for RTT calculation. The address of the memory of a TCP packet is computed based on the hash value of the fid and the expected ACK (eACK) number. If the memory location is available (empty or expired), then the packet could be inserted. Otherwise, a different memory location is checked using another hash function. This process keeps repeating until finding a vacant entry, or four hashes are computed (4 stages). Fig. 25 summarizes the implemented data structure.

A similar data structure adopted by Khooi et al. [66] is the CHT [200] with four hash functions (four logical stages) to implement ACLs. Essentially, in CHT, when a new key collides with an old value using hash h_1, the old key is replaced by the new one. Furthermore, the old key is displaced to a new position based on hash h_2. The process keeps alternating until a vacant entry is found.

Friday et al. [61] claim that botnet-orchestrated DDoS attacks generally target specific vulnerabilities and produce similar signatures. To mitigate this attack, the signature counts are accumulated in the data plane via BF data structure. The memory location of SYN packets is calculated using the CRC_{16} hash of some header fields. Accordingly, the value of the memory will be incremented after every hit. Similarly, to mitigate slow DDoS attacks, the authors used the same technique, such that the timestamps are stored in the memory. As an extension to BF, Bonfim et al. [98] use IBLT, which allows the storage of key-value pairs, to track flows as well as per-flow counters.

Cormode et al. [122] propose a probabilistic data structure, namely CMS, for summarizing data streams stored in the data plane. The data structure can be used in a wide range of important problems. In networking, for instance, CMS is used to detect heavy hitters [179].

The aforementioned approaches deal with memory limitations using data structures and techniques implemented on the switch. Kim et al. [201] present a novel approach to address the memory limitations in the switch by remotely accessing a Dynamic Random Access Memory (DRAM) installed on data center servers comprising Remote Direct Memory Access (RDMA)-capable NICs. The data plane remotely accesses the DRAM through an access channel (RDMA over Converged Ethernet (RoCE)), see Fig. 26. It is worth mentioning that the proposed approach is flexible and cost-effective as it uses existing resources in commodity hardware without adding additional infrastructure costs. Most importantly, the proposed approach does not have a major impact on the performance of the switch as it only incurs 1-2 extra microseconds latency.

In the context of security and memory limitations, several future works could be devised. For instance, existing approaches in heavy hitter detection use approximation, such as sketch-based data structure, to achieve high accuracy while being restricted to the hardware resources. In practice, it is hard to fit large sketches in each switch along the data path while achieving high accuracy (≈99%). Ongoing research work, such as in

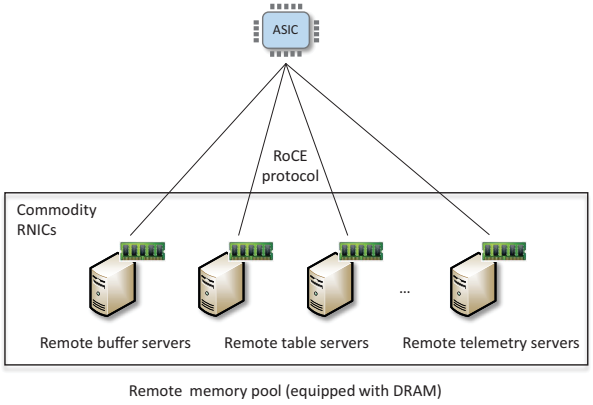

Figure 26: Utilizing the DRAM on commodity servers to expand the data plane memory.

[202], is exploring new data structures that can be fragmented and allocated on multiple hops along the packet's path. Future endeavors could also discover the integration of multiple data structures for efficient DDoS detection, e.g., INDDoS [60] combines Bitmap and CMS, while Jaqen [56] utilizes universal sketches that encompass multiple algorithms.

9.2. Processing Capabilities

Programmable switches process a limited number of primitives, i.e., they cannot perform arbitrary operations. Instead, they support a small set of simple operations. For instance, since division is much slower than addition, the switching hardware usually does not support division. Moreover, floating point arithmetic is not a mandatory feature that is enforced by P4. Also, programmable switches support a limited number of operations per packet in order to operate at line rate. These limitations affect how a number of security applications are implemented in the network.

In attempts to detect attacks in the network, several proposed techniques implement ML in the control plane since programmable switches can not perform complex computations. Moreover, the majority of ML frameworks encompass complex operations and floating point numbers that are not supported within the P4 data plane.

Early implementations of secure and private routing in the data plane use custom-built cryptography functions that did not achieve the highest level of security, such as CRC_{32} hash function. Reasons behind using simple cryptography primitives on programmable switches might be traced to two factors. First, the overwhelming engineering efforts required to implement standardized cryptographic primitives. Second, the high cost that such implementations impose on the limited hardware resources available [36].

9.2.1. Current Initiatives and Future Work

Current initiatives that overcome the computational limitations in the data plane include in-network workarounds (approximation and precomputations), as well as approaches that use external resources (e.g., local CPU or central controller).

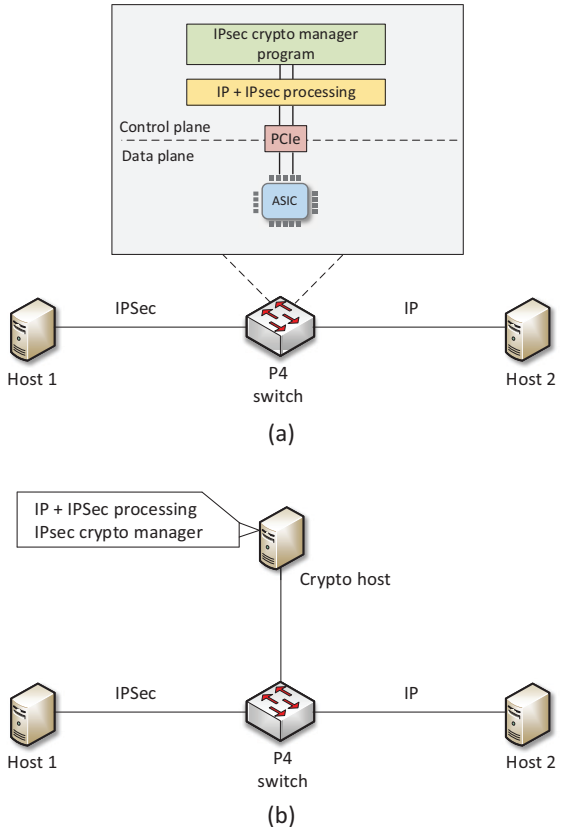

Figure 27: (a) IPsec implementation in the data plane using the switch's ASIC chip. (b) IPsec implementation in the data plane using an external crypto host.

In the context of approximation techniques, designers use the available limited resources on the switch to approximate the desired value, while sacrificing the precision. As an example, Ding et al. [203] propose P4Log and P4Exp algorithms to estimate logarithms and exponential functions, respectively, using only P4-supported arithmetic operations. Based on these algorithms, the authors propose a novel strategy, called P4Entropy, to estimate traffic entropy entirely in the data plane without using the TCAM, nor bounding the number of packets observed.

In the context of precomputation techniques, values are precomputed and stored in match-action tables or registers. For example, Sharma et al. [204] approximate aggregate flow statistics (total number of active flows, as well as sources and destinations communicating through a switch) by designing a cardinality estimator building block. The latter uses the HyperLogLog algorithm that can estimate cardinalities greater than one million with a typical standard error of 2% using 1.5 kilobytes of memory.

Alternatively, Hauser et al. [81] present two workarounds to offload the computationally intensive functions of the IPsec crypto manager. The first approach considers offloading the computations to the internal CPU module connected to the Tofino ASIC via the Peripheral Component Interconnect express (PCIe), i.e., the computations are performed locally on the switch's CPU, see Fig. 27(a). The second approach offloads the IPsec-related flows to an external crypto host, see Fig. 27(b).

Future security frameworks must utilize the current initiatives in processing challenges to perform more operations in the data plane. For instance, explore attacks that require cryptographic data, such as Malformed SSL flood, SIP DDoS [67] with packets (e.g., TLS and Datagram TLS (DTLS)). Also, in P4-based cryptography schemes, future security protocols can be explored, such as TLS, Point-to-Point Protocol (PPP), etc.

9.3. Program Virtualization

In today's network, the needs of a specific business are satisfied via services (e.g., malware detection and prevention [205], location services, etc.) that adhere to policies defining operational characteristics and access control. Inserting such services into the network requires one or more Network Functions (NFs), such as firewall and load balancer, that satisfy the defined policies [206]. Currently, the programmable data plane does not support virtualization, thus, it is challenging to run multiple NFs simultaneously on a single switch while achieving isolation [207]. However, in various cases, operators desire more than one context (i.e., NF) even when there is only one physical switch [208].

The main challenges of program virtualization include resource isolation, performance isolation, and security isolation. Resource isolation ensures that some resources in the switch (e.g., tables, entries, registers, etc.) are dedicated to a specific program (NF). Performance isolation dictates that the execution of one program does not affect the performance of another. Security isolation discusses the access rights and privileges of resources for each program; for instance, packets that are processed by one program cannot access data stored by another program.

9.3.1. Current Initiatives and Future Work

HyPer4 [208] is a portable virtualization solution that aims to develop a P4 program that can be dynamically configured to emulate other programs. HyPer4 can isolate customers and/or equipment (network slicing), store multiple network device configurations (network snapshotting), allow modular development (create several virtual devices, such as a router, and a firewall within the switch), support multiple tenants, and provide standard high-level features.

HyperVDP [209] overcomes a number of limitations in HyPer4, such as performance and excessive usage of hardware resources. HyperVDP defines a 4-byte Description Header (DH) that encapsulates the original packet. The DH includes information such as the length of the packet's header, as a result, HyperVDP parses the whole packet header at once without the need for packet resubmission as in HyPer4.

HyPer4 [208] and HyperVDP [209] consist of a P4 program that emulates the behavior of multiple programs (i.e., emulation-based virtualization). Such approaches feature seamless reconfiguration by allowing different programs to be emulated at runtime. Despite this advantage, emulation-based virtualization techniques exhaust the resources of the switches since P4 programs are translated into table entries; thus, significantly adding overhead as compared to native P4 programs. Furthermore, such approaches do not provide CPU isolation; thus, a disruption in one program can affect the others [180]. Sequeira [180] overcomes emulation-based virtualization disadvantages by adopting code merging technology, in which multiple P4 programs are merged into one monolithic program at compile-time. In particular, the authors extend P4Visor [210] to merge more than two programs while enforcing correctness and isolation.

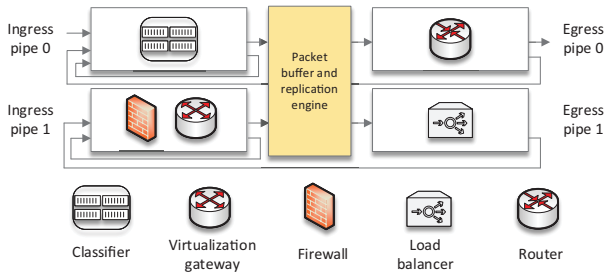

Figure 28: Example of network function placement on pipelines.

Stoyanov et al. [211] propose a Multi Tenant Portable Switch Architecture (MTPSA) that achieves performance, resource, and security isolation. The authors focus on security isolation to prevent attacks, such as infinitely cloning and recirculating packets to create a DDoS attack or information disclosure from packets processed by other user programs. To achieve security isolation, the authors introduce the concept of superuser and user programs within the switch pipeline. The superuser program includes the ingress and egress pipelines that define the pre and post-processing of packets (e.g., assign packets to user programs). User programs are standard P4 programs (identified by a unique id) and share the same superuser ingress and egress pipelines. The superuser assigns roles and privileges for user programs so that each program observes and operates only on its own packets, and each program can execute actions based on the assigned privileges (e.g., no recirculation allowed).

Wu et al. [212] present Dejavu, a system that deploys an entire service chain to a programmable switch. Conceptually, Dejavu uses a customized header format based on the IETF NSH proposal [160] that includes platform metadata (e.g., ingress/egress ports), recirculate flag (whether to recirculate the packet), and others. For efficient placement of NFs on the pipeline, the authors study the tradeoff of sequential and parallel compositions. Sequential composition places NF back-to-back on a single ingress/egress pipe (pipelet), thus, the transition between NFs is cheap but requires a separate match-action unit stage for each NF. Parallel composition places NF side-by-side, thus, allowing for resource sharing but requires recirculation/ resubmission to transition between NF. Fig. 28 shows a prototype of Dejavu with five NFs implemented on Tofino ASIC with two pipelines. In the implemented prototype, the ports corresponding to ingress pipe 1 are set into loopback mode, thus, traffic arriving at ingress pipe 1 comes only from egress pipe 1 (via recirculation). Thus, arriving packets will always pass in the classifier first (i.e., from ingress pipe 0).

Future work in program virtualization could investigate general optimization techniques to efficiently place the NFs in the switch with respect to the available resources. Additionally, for

NFs that cannot be placed in one switch (e.g., advanced DDoS schemes), network-wide function placement can be explored to distribute the functions across multiple switches.

Currently, there is no standard scheme that defines the placement of network devices. For instance, programmable switches can be placed on the network edge to swiftly filter DDoS attacks, firewalls and IDS can be placed inside the network for advanced filtering and monitoring, and SmartNICs can easily employ cryptography between end hosts. The problem of function placement in the network is still an open problem that is yet to be addressed.

9.4. Deep Packet Inspection (DPI)

General-purpose CPUs provide poor performance when handling bulk of data, and they should only handle metadata or a partial amount of the data [213]. The capabilities brought by programmable switches provide promising opportunities to handle data processing and perform DPI. However, implementing DPI in the switch is not trivial since P4 applications need to satisfy high computation, memory, and data transfer requirements. Additionally, the P4 language is designed to be restricted (e.g., no looping constructs) in order to preserve the performance. Accordingly, the majority of P4 applications perform limited inspection over the packet's header. For instance, application layer P4 defenses are limited, and the literature focuses on attacks that can be inferred from traditional header fields (e.g., IP, TCP/UDP, etc.). The same limitations apply to application-level network monitoring, load balancing, and others.

9.4.1. Current Initiatives and Future Work

Jepsen et al. [213] developed a system for locating the occurrences of string keywords in the payload using P4. The PISA-based Parallel Search (PPS) takes a set of search patterns and converts it to partitioned Deterministic Finite Automata (DFAs) that can run on each stage, thus, achieving parallelism across pipelines.

DeepMatch [214] provides a line rate DPI primitive in the data plane using Netronome SMART [215]. DeepMatch showcases new applications in the data plane that were not realized before, such as QoS policies and network monitoring that require processing application layer information. Additionally, IDS can be integrated to perform advanced DPI security policies. DeepMatch provides stateless intra-packet and stateful inter-packet regex matching capabilities, as well as supports reordering of packets since the content specified by a regex may appear anywhere in a flow. Similar to PPS [213], DeepMatch transforms the regex into a DFA and implements it using a state transition table.

Considering the limited memory, a complex regex expression can result in a DFA state explosion. Future work could explore optimizing the regex to occupy less memory in the switch, and consequently be able to perform complex matching operations. Furthermore, future DPI in the context of security could include: malware detection, in which packets belonging to a certain flow and encoding malware-based keywords are identified on the fly; email spam filtering, etc.

9.5. Interoperability

Despite the increasing interest in programmable data planes and the advantages they bring to the network, operators are unlikely to fully change their legacy infrastructure in one shot. This is due to the additional budget costs incurred (e.g., buying new equipment), as well as operational costs (e.g., training new staff, maintenance, etc.). During the development of a P4-based approach, arbitrarily placing the switch in a hybrid network (a network that contains legacy and programmable switches) might seem like a simple solution. However, evaluations in [54] show that the accuracy is largely affected by the position of the programmable switch in the hybrid network. Thus, placing switches blindly is not a good practice.

9.5.1. Current Initiatives and Future Work

In network anonymization, PANEL [77] offers partial deployability and compatibility with legacy networks. In heavy hitter detection, [54] proposes an approach to incrementally select the suitable legacy devices and replace them with programmable switches, while monitoring as many flows as possible. In sketch-based network monitoring, Shi et al. [216] propose an approach to incrementally deploy programmable switches with legacy networks. The authors use two Integer Linear Programming (ILP) models to cover more traffic and improve the accuracy simultaneously.

Future work in incremental deployment should span across all P4 applications. For instance, in security, network-wide coordination for DDoS detection in the presence of legacy devices could be devised. This encourages operators to start incorporating programmability in their network.

9.6. Self-driving Network

Current in-network approaches require manual interaction and analysis from the network operator in order to create new policies and adjust to the changes in the network. This handcrafted method introduces poor scalability and robustness, as well as several burdens on the operator. Examples of operator-required interaction span across all security applications, such as threshold-based adaptation in volumetric attacks (e.g., DDoS defenses and heavy hitter detectors), maintenance of ACLs, and others. Solutions that perform a local control loop of measuring, analyzing, and acting on each in-network device separately are inefficient since they only operate on a fraction of the system's traffic [217].

9.6.1. Current Initiatives and Future Work

A recent work by Mai et al. [217] propose a hybrid in-network intelligence control architecture that relies on ML for self-adaptive learning. The architecture includes a programmable data plane to host control functions (e.g., measurements, firewall, heavy hitter detection). Additionally, an intelligent centralized management plane collects data from the network, learns its behavior, and automatically generates control strategies. Among the use cases evaluated, a Bayesian-based in-network DDoS detection was able to constantly learn defense tactics (Bayesian classifier) in a distributed and automated fashion.

Future security solutions could explore more ML techniques to make the approach more intelligent, resilient against network changes, and require minimal user interaction. For instance, approaches in heavy hitter detection, such as in [54], are ineffective when the routing and flow statistics are altered. An enhancement for such approaches is to develop a self-driving network that can automatically readjust the data plane configuration when changes in the system occur.

Challenges introduced by self-driving networks include communication overhead between the intelligent controller and the switches; latency introduced by the controller running the ML module; saturation attacks on a centralized controller. This could be critical for time-sensitive security solutions and opens new horizons for future optimization approaches.

10. Conclusion

This work presents a survey on the security of programmable data plane, specifically those utilizing the P4 language. First, the survey gives some background information about the P4 language and the PSA, as well as the specifications, advantages, and limitations of programmable switches. Then, state-of-the-art related works are presented. Motivated by the lack of an inclusive P4 security-centric work, the survey provides a taxonomy of P4 security implementations while categorizing them under three fundamental security objectives (network availability, anonymity and confidentiality, and operational security provisioning). Afterward, the survey performs a STRIDE-based vulnerability assessment on driving P4 applications (e.g., network telemetry, load-balancing, congestion control, etc.) and proposes a number of remediation techniques. Finally, the survey concludes with a discussion about the current challenges and initiatives in programmable devices, as well as pinpoints potential future work and open research areas.

Acknowledgment

This material is based upon work supported by the National Science Foundation under grant numbers 2118311, 2104273, and 1925484, funded by the Office of Advanced Cyberinfrastructure (OAC).

Table 15: Abbreviations Table

Abbreviation	Term
5G	Fifth-Generation
ACK	ACKnowledgment
ACL	Access Control List
AES	Advanced Encryption Standard
AES-CTR	AES Counter Mode
AES-GCM	AES in Galois/Counter Mode
ALU	Arithmetic Logic Unit
AM-PM	Alternate Marking-Performance Measurement
ANT	Active Network Telemetry
API	Application Programming Interface
AR-DDoS	Amplified Reflection DDoS
ARP	Address Resolution Protocol
AS	Autonomous System
ASIC	Application-Specific Integrated Circuit
BF	Bloom Filter
BMv2	Behavioral Model
BNF	Backus-Naur Form
BNN	Binarized Neural Network
BYOD	Bring Your Own Device
CMS	Count-Min Sketch
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CTL	Computation Tree Logic
DAD	Duplicate Address Detection
DDoS	Distributed Denial of Service
DFA	Deterministic Finite Automata
DH	Description Header
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Server
DNSSEC	DNS Security Extensions
DoS	Denial of Service
DPI	Deep Packet Inspection
DRAM	Dynamic Random Access Memory
DTLS	Datagram TLS
eACK	expected ACKnowledgement
ECN	Explicit Congestion Notification
EFSM	Extended Finite State Machine
ESP	Encapsulating Security Payload
fid	flow id
FMA	Flow Marker Accumulator
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
Gbps	Gigabits per second
GCL	Guarded Command Language
GPRS	General Packet Radio Service
GPS	Global Positioning System
GTP	GPRS Tunneling Protocol
HCF	Hop Count Filtering
HMAC	Hash-based Message Authentication Code
IBLT	Invertible Bloom Lookup Table
IDS	Intrusion Detection System
IKE	Internet Key Exchange
ILP	Integer Linear Programming
INT	In-band Network Telemetry
IOAM	In-situ Operation Administration
IoT	Internet of Things
IP	Internet Protocol
IP2HC	IP-to-Hop-Count
IPsec	Internet Protocol Security
ISP	Internet Service Provider
IXP	Internet eXchange Point
LPM	Longest Prefix Match
LTE EPC	Long Term Evolution Evolved Packet Core
MAC	Media Access Control
MACsec	MAC security
ML	Machine Learning
MPLS	Multiprotocol Label Switching
NA	Neighbor Advertisement

Abbreviation	Term
NF	Network Function
NFV	Network Function Virtualization
NGFW	Next-Generation Fire Wall
NIF	Network Ingress Filtering
NS	Neighbor Solicitation
NSH	Network Service Header
OTP	One-time password
P4	Programming Protocol-Independent Packet Processors
PBT	Postcard-Based Telemetry
PCIe	Peripheral Component Interconnect express
PII	Personally Identifiable Information
PPM	Packet Processing Modules
PPP	Point-to-Point Protocol
PSA	Portable Switch Architecture
QoS	Quality of Service
RDMA	Remote Direct Memory Access
RISC	Reduced Instruction Set Computer
RMT	Reconfigurable Match table
RoCE	RDMA over Converged Ethernet
RPF-Feasible	Feasible Reverse Path Forwarding
RPF-Loose	Loose Reverse Path Forwarding
RPF-Strict	Strict Reverse Path Forwarding
SA	Security Association
SAVI	Source Address Validation Improvement
SDN	Software-Defined Networking
SFC	Service Function Chaining
SIP	Session Initiation Protocol
SMT	Satisfiability Modulo Theory
SmartNICs	Smart Network Interface Cards
SPM	Spoofing Prevention Method
SRAM	Static Random-Access Memory
SSL	Secure Socket Layer
SYN	SYNchronize
Tbps	Terabits per second
TCAM	Ternary Content Addressable Memory
TCP	Transport Control Protocol
TLS	Transport Layer Security
TTL	Time To Live
UDP	User Datagram Protocol
vEPG	virtual Evolved Packet Gateway
VHE	Vector Homomorphic Encryption
VLIW	Very Long Instruction Word
VPN	Virtual Private Network
VXLAN	Virtual Extensible Local Area Network
XFSM	eXtended Finite State Machines

References

[1] 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 44 (3) (2014) 87–95.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, ACM SIGCOMM Computer Communication Review 38 (2) (2008) 69–74.

[3] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, M. Horowitz, Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN, ACM SIGCOMM Computer Communication Review 43 (4) (2013) 99–110.

[4] Open Networking Foundation, Openflow switch specification version 1.5.1, [Online]. Available: `https://tinyurl.com/4hepckyu`.

[5] G. Bianchi, M. Bonola, A. Capone, C. Cascone, Openstate: programming platform-independent stateful openflow applications inside the switch, ACM SIGCOMM Computer Communication Review 44 (2) (2014) 44–51.

[6] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, J. Wu, Poseidon: Mitigating volumetric DDoS attacks with programmable switches, in: Proceedings of NDSS, 2020.

[7] T. P. Morgan, Vmware, cisco stretch virtual lans across the heavens, [Online]. Available: `https://tinyurl.com/p72a9d77`.

[8] M. Mahalingam, D. G. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, C. Wright, Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks., RFC 7348 (2014) 1–22.

[9] T. Benson, A. Akella, A. Shaikh, S. Sahu, Cloudnaas: a cloud networking platform for enterprise applications, in: Proceedings of the 2nd ACM Symposium on Cloud Computing, 2011, pp. 1–13.

[10] S. K. Fayaz, Y. Tobioka, V. Sekar, M. Bailey, Bohatei: Flexible and elastic DDoS defense, in: 24th USENIX Security Symposium (USENIX Security 15), 2015, pp. 817–832.

[11] A. Panchenko, L. Pimenidis, J. Renner, Performance analysis of anonymous communication channels provided by tor, in: 2008 Third International Conference on Availability, Reliability and Security, IEEE, 2008, pp. 221–228.

[12] R. Dingledine, S. J. Murdoch, Performance improvements on tor or, why tor is slow and what we're going to do about it, Online: http://www.torproject.org/press/presskit/2009-03-11-performance.pdf (2009).

[13] A. AlSabeh, E. Kfoury, J. Crichigno, E. Bou-Harb, Leveraging sonic functionalities in disaggregated network switches, in: 2020 43rd International Conference on Telecommunications and Signal Processing (TSP), IEEE, 2020, pp. 457–460.

[14] R. Miao, H. Zeng, C. Kim, J. Lee, M. Yu, Silkroad: Making stateful layer-4 load balancing fast and cheap using switching ASICs, in: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, 2017, pp. 15–28.

[15] R. Ricart-Sanchez, P. Malagon, J. M. Alcaraz-Calero, Q. Wang, Hardware-accelerated firewall for 5G mobile networks, in: 2018 IEEE 26th International Conference on Network Protocols (ICNP), IEEE, 2018, pp. 446–447.

[16] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, I. Stoica, Netcache: Balancing key-value stores with fast in-network caching, in: Proceedings of the 26th Symposium on Operating Systems Principles, 2017, pp. 121–136.

[17] A. Satapathy, Comprehensive study of P4 programming language and software-defined networks, Ph.D. thesis, Institute for Development and Research in Banking Technology (2018).

[18] E. Kaljic, A. Maric, P. Njemcevic, M. Hadzialic, A survey on data plane flexibility and programmability in software-defined networking, IEEE Access 7 (2019) 47804–47840.

[19] W. L. da Costa Cordeiro, J. A. Marques, L. P. Gaspary, Data plane programmability beyond openflow: Opportunities and challenges for network and service operations and management, Journal of Network and Systems Management 25 (4) (2017) 784–818.

[20] R. Bifulco, G. Rétvári, A survey on the programmable data plane: Abstractions, architectures, and open problems, in: 2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR), IEEE, 2018, pp. 1–7.

[21] H. Stubbe, P4 compiler & interpreter: A survey, Future Internet (FI) and Innovative Internet Technologies and Mobile Communication (IITM) 47 (2017).

[22] A.-A. Agape, M. C. Danceanu, R. R. Hansen, S. Schmid, Charting the security landscape of programmable dataplanes, arXiv preprint arXiv:1807.00128 (2018).

[23] M. V. Dumitru, D. Dumitrescu, C. Raiciu, Can we exploit buggy P4 programs?, in: Proceedings of the Symposium on SDN Research, 2020, pp. 62–68.

[24] C. Black, S. Scott-Hayward, A survey on the verification of adversarial

data planes in software-defined networks, in: Proceedings of the 2021 ACM International Workshop on Software Defined Networks & Network Function Virtualization Security, 2021, pp. 3–10.

[25] O. Michel, R. Bifulco, G. Rétvári, S. Schmid, The programmable data plane: Abstractions, architectures, algorithms, and applications, ACM Computing Surveys (CSUR) 54 (4) (2021) 1–36.

[26] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, M. Menth, A survey on data plane programming with P4: Fundamentals, advances, and applied research, arXiv preprint arXiv:2101.10632 (2021).

[27] S. Kaur, K. Kumar, N. Aggarwal, A review on P4-programmable data planes: Architecture, research efforts, and future directions, Computer Communications (2021).

[28] E. F. Kfoury, J. Crichigno, E. Bou-Harb, An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends, IEEE Access (2021).

[29] S. Hernan, S. Lambert, T. Ostwald, A. Shostack, Threat modeling-uncover security design flaws using the stride approach, MSDN Magazine-Louisville (2006) 68–75.

[30] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, M. Conti, A survey on the security of stateful SDN data planes, IEEE Communications Surveys & Tutorials 19 (3) (2017) 1701–1725.

[31] P4lang, P4 language consortium. behavioral model (BMv2), [Online]. Available: `https://github.com/p4lang/behavioral-model` (2020).

[32] S. Ibanez, G. Brebner, N. McKeown, N. Zilberman, The P4-> NetFPGA workflow for line-rate packet processing, in: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2019, pp. 1–9.

[33] Intel Tofino, P4-programmable ethernet switch ASIC that delivers better performance at lower power, [Online]. Available: `https://tinyurl.com/3mh96jns`.

[34] E. Bou-Harb, N.-E. Lakhdari, H. Binsalleeh, M. Debbabi, Multidimensional investigation of source port 0 probing, Digital Investigation 11 (2014) S114–S123.

[35] The P4.org Architecture Working Group, P4_16 portable switch architecture (PSA), [Online]. Available: `https://tinyurl.com/zyuu8xtd` (2021).

[36] X. Chen, Implementing AES encryption on programmable switches via scrambled lookup tables, in: Proceedings of the Workshop on Secure Programmable Network Infrastructure, 2020, pp. 8–14.

[37] V. Gurevich, P4 mapping to Barefoot Tofino(tm), [Online]. Available: `https://tinyurl.com/4rax6rvf`.

[38] N. McKeown, End-to-end programmable forwarding, [Online]. Available: `https://tinyurl.com/uj6uptkw` (2020).

[39] P. Guide, Intel® 64 and ia-32 architectures software developer's manual, Volume 3B: System programming Guide, Part 2 (11) (2011).

[40] N. McKeown, Why does the Internet need a programmable forwarding plane, [Online]. Available: `https://tinyurl.com/zpsv4waz` (2017).

[41] Arista, Arista 7170 Multi-function Programmable Networking, [Online]. Available: `https://tinyurl.com/3amndwjy`.

[42] E. Networks, Wedge 100BF-32X 100GbE Data Center Switch, [Online]. Available: `https://tinyurl.com/z6vad83c`.

[43] R. Ben-Basat, X. Chen, G. Einziger, O. Rottenstreich, Efficient measurement on programmable switches using probabilistic recirculation, in: 2018 IEEE 26th International Conference on Network Protocols (ICNP), IEEE, 2018, pp. 313–323.

[44] H. Soni, M. Rifai, P. Kumar, R. Doenges, N. Foster, Composing dataplane programs with μP4, in: Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, 2020, pp. 329–343.

[45] G. Li, M. Zhang, C. Liu, X. Kong, A. Chen, G. Gu, H. Duan, Nethcf: Enabling line-rate and adaptive spoofed IP traffic filtering, in: 2019 IEEE 27th international conference on network protocols (ICNP), IEEE, 2019, pp. 1–12.

[46] G. Simsek, H. Bostan, A. K. Sarica, E. Sarikaya, A. Keles, P. Angin, H. Alemdar, E. Onur, Dropppp: A P4 approach to mitigating DoS attacks in SDN, in: International Workshop on Information Security Applications, Springer, 2019, pp. 55–66.

[47] P. Kuang, Y. Liu, L. He, P4DAD: Securing duplicate address detection using P4, in: ICC 2020-2020 IEEE International Conference on Communications (ICC), IEEE, 2020, pp. 1–7.

[48] N. Narayanan, G. C. Sankaran, K. M. Sivalingam, Mitigation of security attacks in the SDN data plane using P4-enabled switches, in: 2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), IEEE, 2019, pp. 1–6.

[49] H. Gondaliya, G. C. Sankaran, K. M. Sivalingam, Comparative evaluation of IP address anti-spoofing mechanisms using a P4/NetFPGA-based switch, in: Proceedings of the 3rd P4 Workshop in Europe, 2020, pp. 1–6.

[50] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, J. Rexford, Heavy-hitter detection entirely in the data plane, in: Proceedings of the Symposium on SDN Research, 2017, pp. 164–176.

[51] L. Tang, Q. Huang, P. P. Lee, A fast and compact invertible sketch for network-wide heavy flow detection, IEEE/ACM Transactions on Networking 28 (5) (2020) 2350–2363.

[52] J. Kučera, D. A. Popescu, G. Antichi, J. Kořenek, A. W. Moore, Seek and push: Detecting large traffic aggregates in the dataplane, arXiv preprint arXiv:1805.05993 (2018).

[53] R. Harrison, Q. Cai, A. Gupta, J. Rexford, Network-wide heavy hitter detection with commodity switches, in: Proceedings of the Symposium on SDN Research, 2018, pp. 1–7.

[54] D. Ding, M. Savi, G. Antichi, D. Siracusa, An incrementally-deployable P4-enabled architecture for network-wide heavy-hitter detection, IEEE Transactions on Network and Service Management 17 (1) (2020) 75–88.

[55] G. Li, M. Zhang, S. Wang, C. Liu, M. Xu, A. Chen, H. Hu, G. Gu, Q. Li, J. Wu, Enabling performant, flexible and cost-efficient DDoS defense with programmable switches, IEEE/ACM Transactions on Networking (2021).

[56] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, V. Sekar, Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric DDoS attacks with programmable switches.

[57] Â. C. Lapolli, J. A. Marques, L. P. Gaspary, Offloading real-time DDoS attack detection to programmable data planes, in: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), IEEE, 2019, pp. 19–27.

[58] A. da Silveira Ilha, Â. C. Lapolli, J. A. Marques, L. P. Gaspary, Euclid: A fully in-network, P4-based approach for real-time DDoS attack detection and mitigation, IEEE Transactions on Network and Service Management (2020).

[59] M. Dimolianis, A. Pavlidis, V. Maglaris, A multi-feature DDoS detection schema on P4 network hardware, in: 2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), IEEE, 2020, pp. 1–6.

[60] D. Ding, M. Savi, F. Pederzolli, M. Campanella, D. Siracusa, In-network volumetric DDoS victim identification using programmable commodity switches, IEEE Transactions on Network and Service Management (2021).

[61] K. Friday, E. Kfoury, E. Bou-Harb, J. Crichigno, Towards a unified in-network DDoS detection and mitigation strategy, in: submitted to IEEE Conference on Network Softwarization (NetSoft), Ghent, Belgium, 2020.

[62] F. Musumeci, V. Ionata, F. Paolucci, F. Cugini, M. Tornatore, Machine-learning-assisted DDoS attack detection with P4 language, in: ICC 2020-2020 IEEE International Conference on Communications (ICC), IEEE, 2020, pp. 1–6.

[63] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, P. Castoldi, P4 edge node enabling stateful traffic engineering and cyber security, Journal of Optical Communications and Networking 11 (1) (2019) A84–A95.

[64] D. Scholz, S. Gallenmüller, H. Stubbe, B. Jaber, M. Rouhi, G. Carle, Me love (SYN-) cookies: SYN flood mitigation in programmable data planes, arXiv preprint arXiv:2003.03221 (2020).

[65] M. Kuka, K. Vojanec, J. Kučera, P. Benáček, Accelerated DDoS attacks mitigation using programmable data plane, in: 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), IEEE, 2019, pp. 1–3.

[66] X. Z. Khooi, L. Csikor, D. M. Divakaran, M. S. Kang, Dida: Distributed in-network defense architecture against amplified reflection DDoS at-

tacks, in: 2020 6th IEEE Conference on Network Softwarization (NetSoft), IEEE, 2020, pp. 277–281.
[67] A. Febro, H. Xiao, J. Spring, Distributed SIP DDoS defense with P4, in: 2019 IEEE Wireless Communications and Networking Conference (WCNC), IEEE, 2019, pp. 1–8.
[68] A. Kheradmand, G. Rosu, P4k: A formal semantics of P4 and applications, arXiv preprint arXiv:1804.01468 (2018).
[69] R. Stoenescu, D. Dumitrescu, M. Popovici, L. Negreanu, C. Raiciu, Debugging P4 programs with vera, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, 2018, pp. 518–532.
[70] D. Dumitrescu, R. Stoenescu, L. Negreanu, C. Raiciu, bf4: towards bug-free P4 programs, in: Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, 2020, pp. 571–585.
[71] J. Liu, W. Hallahan, C. Schlesinger, M. Sharif, J. Lee, R. Soulé, H. Wang, C. Caşcaval, N. McKeown, N. Foster, P4v: Practical verification for programmable data planes, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, 2018, pp. 490–503.
[72] L. Freire, M. Neves, L. Leal, K. Levchenko, A. Schaeffer-Filho, M. Barcellos, Uncovering bugs in P4 programs with assertion-based verification, in: Proceedings of the Symposium on SDN Research, 2018, pp. 1–7.
[73] F. Ruffy, T. Wang, A. Sivaraman, Gauntlet: Finding bugs in compilers for programmable packet processing, arXiv preprint arXiv:2006.01074 (2020).
[74] A. Nötzli, J. Khan, A. Fingerhut, C. Barrett, P. Athanas, P4pktgen: Automated test case generation for P4 programs, in: Proceedings of the Symposium on SDN Research, 2018, pp. 1–7.
[75] A. Shukla, K. N. Hudemann, A. Hecker, S. Schmid, Runtime verification of P4 switches with reinforcement learning, in: Proceedings of the 2019 Workshop on Network Meets AI & ML, 2019, pp. 1–7.
[76] H. Kim, A. Gupta, Ontas: Flexible and scalable online network traffic anonymization system, in: Proceedings of the 2019 Workshop on Network Meets AI & ML, 2019, pp. 15–21.
[77] H. M. Moghaddam, A. Mosenia, Anonymizing masses: Practical lightweight anonymity at the network level, arXiv preprint arXiv:1911.09642 (2019).
[78] T. Datta, N. Feamster, J. Rexford, L. Wang, SPINE: Surveillance protection in the network elements, in: 9th USENIX Workshop on Free and Open Communications on the Internet (FOCI 19), 2019.
[79] L. Wang, H. Kim, P. Mittal, J. Rexford, Programmable in-network obfuscation of dns traffic, in: NDSS: DNS Privacy Workshop, 2021.
[80] F. Hauser, M. Schmidt, M. Häberle, M. Menth, P4-MACsec: Dynamic topology monitoring and data layer protection with MACsec in P4-based SDN, IEEE Access 8 (2020) 58845–58858.
[81] F. Hauser, M. Häberle, M. Schmidt, M. Menth, P4-IPsec: Implementation of IPsec gateways in P4 with SDN control for host-to-site scenarios, arXiv preprint arXiv:1907.03593 (2019).
[82] G. Liu, W. Quan, N. Cheng, D. Gao, N. Lu, H. Zhang, X. Shen, Softwarized IoT network immunity against eavesdropping with programmable data planes, IEEE Internet of Things Journal (2021).
[83] D. Scholz, A. Oeldemann, F. Geyer, S. Gallenmüller, H. Stubbe, T. Wild, A. Herkersdorf, G. Carle, Cryptographic hashing in P4 data planes, in: 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), IEEE, 2019, pp. 1–6.
[84] Y.-B. Lin, T.-J. Huang, S.-C. Tsai, Enhancing 5G/IoT transport security through content permutation, IEEE Access 7 (2019) 94293–94299.
[85] L. Malina, D. Smekal, S. Ricci, J. Hajny, P. Cíbik, J. Hrabovsky, Hardware-accelerated cryptography for software-defined networks with P4, in: International Conference on Information Technology and Communications Security, Springer, 2020, pp. 271–287.
[86] R. Datta, S. Choi, A. Chowdhary, Y. Park, P4guard: Designing P4 based firewall, in: MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM), IEEE, 2018, pp. 1–6.
[87] J. Cao, J. Bi, Y. Zhou, C. Zhang, Cofilter: A high-performance switch-assisted stateful packet filter, in: Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos, 2018, pp. 9–11.
[88] J. Li, H. Jiang, W. Jiang, J. Wu, W. Du, SDN-based stateful firewall for cloud, in: 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), IEEE, 2020, pp. 157–161.
[89] A. Almaini, A. Al-Dubai, I. Romdhani, M. Schramm, Delegation of authentication to the data plane in software-defined networks, in: 2019 IEEE International Conferences on Ubiquitous Computing & Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS), IEEE, 2019, pp. 58–65.
[90] A. Almaini, A. Al-Dubai, I. Romdhani, M. Schramm, A. Alsarhan, Lightweight edge authentication for software defined networks, Computing 103 (2) (2021) 291–311.
[91] E. O. Zaballa, D. Franco, Z. Zhou, M. S. Berger, P4knocking: Offloading host-based firewall functionalities to the network, in: 2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), IEEE, 2020, pp. 7–12.
[92] R. Ricart-Sanchez, P. Malagon, J. M. Alcaraz-Calero, Q. Wang, NetFPGA-based firewall solution for 5G multi-tenant architectures, in: 2019 IEEE International Conference on Edge Computing (EDGE), IEEE, 2019, pp. 132–136.
[93] J. Xing, Q. Kang, A. Chen, Netwarden: Mitigating network covert channels while preserving performance, in: 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 2039–2056.
[94] Q. Kang, L. Xue, A. Morrison, Y. Tang, A. Chen, X. Luo, Programmable in-network security for context-aware BYOD policies, in: Proc. USENIX Security, 2020.
[95] J. Xing, W. Wu, A. Chen, Architecting programmable data plane defenses into the network with fastflex, in: Proceedings of the 18th ACM Workshop on Hot Topics in Networks, 2019, pp. 161–169.
[96] A. Laraba, J. François, I. Chrisment, S. R. Chowdhury, R. Boutaba, Defeating protocol abuse with P4: Application to explicit congestion notification, in: 2020 IFIP Networking Conference (Networking), IEEE, 2020, pp. 431–439.
[97] A. Laraba, J. François, S. R. Chowdhury, I. Chrisment, R. Boutaba, Mitigating TCP protocol misuse with programmable data planes, IEEE Transactions on Network and Service Management 18 (1) (2021) 760–774.
[98] M. Bonfim, M. Santos, K. Dias, S. Fernandes, A real-time attack defense framework for 5G network slicing, Software: Practice and Experience (2020).
[99] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, A. Madeira, Flowlens: Enabling efficient flow classification for ml-based network security applications, in: Proceedings of the 28th Network and Distributed System Security Symposium (San Diego, CA, USA, 2021.
[100] Q. Qin, K. Poularakis, K. K. Leung, L. Tassiulas, Line-speed and scalable intrusion detection at the network edge via federated learning, in: 2020 IFIP Networking Conference (Networking), IEEE, 2020, pp. 352–360.
[101] M. Majkowski, The real cause of large DDoS - IP spoofing, [Online]. Available: `https://tinyurl.com/yktw6ud2`.
[102] CAIDA, Spoofer, [Online]. Available: `https://www.caida.org/projects/spoofer/`.
[103] P. Ferguson, D. Senie, rfc2827: network ingress filtering: defeating denial of service attacks which employ IP source address spoofing (2000).
[104] F. Baker, P. Savola, Ingress filtering for multihomed networks, Tech. rep., BCP 84, RFC 3704, March (2004).
[105] A. Bremler-Barr, H. Levy, Spoofing prevention method, in: Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies., Vol. 1, IEEE, 2005, pp. 536–547.
[106] J. Bi, J. Wu, G. Yao, F. Baker, Source address validation improvement (SAVI) solution for DHCP, RFC 7513 (2015).
[107] J. Li, M. Sung, J. Xu, L. Li, Large-scale IP traceback in high-speed internet: Practical techniques and theoretical foundation, in: IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004, IEEE, 2004, pp. 115–129.
[108] S. Savage, D. Wetherall, A. Karlin, T. Anderson, Practical network support for IP traceback, in: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, 2000, pp. 295–306.

[109] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, W. T. Strayer, Hash-based IP traceback, ACM SIGCOMM Computer Communication Review 31 (4) (2001) 3–14.

[110] A. D. Keromytis, V. Misra, D. Rubenstein, Sos: Secure overlay services, ACM SIGCOMM Computer Communication Review 32 (4) (2002) 61–72.

[111] J. Li, J. Mirkovic, M. Wang, P. Reiher, L. Zhang, Save: Source address validity enforcement protocol, in: Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 3, IEEE, 2002, pp. 1557–1566.

[112] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, S. Shenker, Controlling high bandwidth aggregates in the network, ACM SIGCOMM Computer Communication Review 32 (3) (2002) 62–73.

[113] O. Spatscheck, L. L. Peterson, Defending against denial of service attacks in scout, in: OSDI, Vol. 99, 1999, pp. 59–72.

[114] X. Qie, R. Pang, L. Peterson, Defensive programming: Using an annotation toolkit to build DoS-resistant software, ACM SIGOPS Operating Systems Review 36 (SI) (2002) 45–60.

[115] G. Banga, P. Druschel, J. C. Mogul, Resource containers: A new facility for resource management in server systems, in: OSDI, Vol. 99, 1999, pp. 45–58.

[116] X. Wang, M. K. Reiter, Defending against denial-of-service attacks with puzzle auctions, in: 2003 Symposium on Security and Privacy, 2003., IEEE, 2003, pp. 78–92.

[117] A. Juels, Client puzzles: A cryptographic countermeasure against connection depletion attacks, in: Proc. Networks and Distributed System Security Symposium (NDSS), 1999, 1999.

[118] Introduction to cisco ios netflow - a technical overview, [Online]. Available: `https://tinyurl.com/52jcc282`.

[119] Y. Zhang, S. Singh, S. Sen, N. Duffield, C. Lund, Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications, in: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement, 2004, pp. 101–114.

[120] R. S. Boyer, J. S. Moore, Mjrty—a fast majority vote algorithm, in: Automated Reasoning, Springer, 1991, pp. 105–117.

[121] C. E. Shannon, A mathematical theory of communication, ACM SIGMOBILE mobile computing and communications review 5 (1) (2001) 3–55.

[122] G. Cormode, S. Muthukrishnan, An improved data stream summary: the count-min sketch and its applications, Journal of Algorithms 55 (1) (2005) 58–75.

[123] C. Estan, G. Varghese, M. Fisk, Bitmap algorithms for counting active flows on high speed links, in: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement, 2003, pp. 153–166.

[124] J. Heinanen, R. Guérin, A two rate three color marker, Tech. rep., RFc 2698, SepTeMbeR (1999).

[125] P. Flajolet, É. Fusy, O. Gandouet, F. Meunier, Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm, in: Discrete Mathematics and Theoretical Computer Science, Discrete Mathematics and Theoretical Computer Science, 2007, pp. 137–156.

[126] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, V. Braverman, One sketch to rule them all: Rethinking network flow monitoring with univmon, in: Proceedings of the 2016 ACM SIGCOMM Conference, 2016, pp. 101–114.

[127] A. Jean-Philippe, D. Bernstein, Siphash: a fast short-input prf, Progress in Cryptology-INDOCRYPT (2012) 489–508.

[128] D. Bogdanas, G. Roşu, K-java: a complete semantics of java, in: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2015, pp. 445–456.

[129] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, H. Veith, Model checking, MIT press, 2018.

[130] R. Stoenescu, M. Popovici, L. Negreanu, C. Raiciu, Symnet: Scalable symbolic execution for modern networks, in: Proceedings of the 2016 ACM SIGCOMM Conference, 2016, pp. 314–327.

[131] V. Altukhov, V. Podymov, V. Zakharov, E. Chemeritskiy, Vermont-a toolset for checking SDN packet forwarding policies on-line, in: 2014 International Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), IEEE, 2014, pp. 1–6.

[132] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, S. T. King, Debugging the data plane with anteater, ACM SIGCOMM Computer Communication Review 41 (4) (2011) 290–301.

[133] P. Kazemian, G. Varghese, N. McKeown, Header space analysis: Static checking for networks, in: 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), 2012, pp. 113–126.

[134] A. Khurshid, X. Zou, W. Zhou, M. Caesar, P. B. Godfrey, Veriflow: Verifying network-wide invariants in real time, in: 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), 2013, pp. 15–27.

[135] A. Bogdanov, L. R. Knudsen, G. Leander, F.-X. Standaert, J. Steinberger, E. Tischhauser, Key-alternating ciphers in a provable setting: Encryption using a small number of public permutations, in: Annual international conference on the theory and applications of cryptographic techniques, Springer, 2012, pp. 45–62.

[136] tcpanon, [Online]. Available: `http://netweb.ing.unibs.it/~ntw/tools/tcpanon/`.

[137] T. Gamer, C. Mayer, M. Schöller, Pktanon–a generic framework for profile-based traffic anonymization (2008).

[138] M. G. Reed, P. F. Syverson, D. M. Goldschlag, Anonymous connections and onion routing, IEEE Journal on Selected areas in Communications 16 (4) (1998) 482–494.

[139] J. Fan, J. Xu, M. H. Ammar, S. B. Moon, Prefix-preserving IP address anonymization: Measurement-based security evaluation and a new cryptography-based scheme, Computer Networks 46 (2) (2004) 253–272.

[140] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, E. Roback, Report on the development of the advanced encryption standard (AES), Journal of Research of the National Institute of Standards and Technology 106 (3) (2001) 511.

[141] 802.1AE: MAC Security (MACsec), [Online]. Available: `https://1.ieee802.org/security/802-1ae/`.

[142] S. Kent, R. Atkinson, Security architecture for the internet protocol (1998).

[143] J. Salowey, A. Choudhury, D. McGrew, AES galois counter mode (GCM) cipher suites for TLS, Request for Comments 5288 (2008).

[144] D. J. Bernstein, The Poly1305-AES message-authentication code, in: International Workshop on Fast Software Encryption, Springer, 2005, pp. 32–49.

[145] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, C. Winnerlein, Blake2: simpler, smaller, fast as md5, in: International Conference on Applied Cryptography and Network Security, Springer, 2013, pp. 119–135.

[146] P. Vörös, D. Horpácsi, R. Kitlei, D. Leskó, M. Tejfel, S. Laki, T4p4s: A target-independent compiler for protocol-independent packet processors, in: 2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR), IEEE, 2018, pp. 1–8.

[147] Nfp-4000 theory of operation, [Online]. Available: `https://tinyurl.com/5fpa4j65`.

[148] Siphash, [Online]. Available: `https://tinyurl.com/2ut57b8n`.

[149] Sha3, [Online]. Available: `https://github.com/freecores/sha3`.

[150] Pensando, Pensando announces p4-programmable platform and joins p4 community, [Online]. Available: `https://tinyurl.com/6bwuyeyp` (2021).

[151] Z. Ni, G. Liu, D. Afanasev, T. Wood, J. Hwang, Advancing network function virtualization platforms with programmable nics, in: 2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), IEEE, 2019, pp. 1–6.

[152] L. Lamport, Password authentication with insecure communication, Communications of the ACM 24 (11) (1981) 770–772.

[153] GPRS Tunneling Protocol (GTP) Overview, [Online]. Available: `https://tinyurl.com/4up23wm9`.

[154] H. Hu, G.-J. Ahn, W. Han, Z. Zhao, Towards a reliable SDN firewall, in: Open Networking Summit 2014 (ONS 2014), 2014.

[155] S. Zhang, An overview of network slicing for 5G, IEEE Wireless Communications 26 (3) (2019) 111–117.

[156] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1, arXiv preprint arXiv:1602.02830 (2016).

[157] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, D. Bacon, Federated learning: Strategies for improving communication efficiency, arXiv preprint arXiv:1610.05492 (2016).

[158] M. Ghasemi, T. Benson, J. Rexford, Dapper: Data plane performance diagnosis of TCP, in: Proceedings of the Symposium on SDN Research, 2017, pp. 61–74.

[159] J. Halpern, C. Pignataro, et al., Service function chaining (SFC) architecture, in: RFC 7665, 2015.
[160] P. Quinn, U. Elzur, C. Pignataro, Network service header (NSH), in: RFC 8300, RFC Editor, 2018.
[161] S. R. Chowdhury, M. F. Bari, R. Ahmed, R. Boutaba, Payless: A low cost network monitoring framework for software defined networks, in: 2014 IEEE Network Operations and Management Symposium (NOMS), IEEE, 2014, pp. 1–9.
[162] M. Yu, L. Jose, R. Miao, Software defined traffic measurement with opensketch, in: 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), 2013, pp. 29–42.
[163] H. Song, T. Zhou, Z. Li, J. Shin, K. Lee, Postcard-based on-path flow data telemetry draft-song-ippm-postcard-based-telemetry-06, Tech. rep. (2020).
[164] Q. Kang, J. Xing, A. Chen, Automated attack discovery in data plane systems, in: 12th USENIX Workshop on Cyber Security Experimentation and Test (CSET 19), 2019.
[165] X. Pan, S. Tang, S. Liu, J. Kong, X. Zhang, D. Hu, J. Qi, Z. Zhu, Privacy-preserving multilayer in-band network telemetry and data analytics: For safety, please do not report plaintext data, Journal of Lightwave Technology (2020).
[166] S. Ariyapperuma, C. J. Mitchell, Security vulnerabilities in DNS and DNSSEC, in: The Second International Conference on Availability, Reliability and Security (ARES'07), IEEE, 2007, pp. 335–342.
[167] N. Shevchenko, T. A. Chick, P. O'Riordan, T. P. Scanlon, C. Woody, Threat modeling: a summary of available methods, Tech. rep., Carnegie Mellon University Software Engineering Institute (2018).
[168] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, L. J. Wobker, In-band network telemetry via programmable dataplanes, in: ACM SIGCOMM, 2015.
[169] F. Brockners, S. Bhandari, S. Dara, C. Pignataro, H. Gredler, J. Leddy, S. Youell, D. Mozes, T. Mizrahi, P. Lapukhov, et al., Requirements for in-situ oam, in: Working Draft, Internet-Draft draft-brockners-inband-oam-requirements-03, 2017.
[170] G. Fioccola, A. Capello, M. Cociglio, L. Castaldelli, M. G. Chen, L. Zheng, G. Mirsky, T. Mizrahi, Alternate-marking method for passive and hybrid performance monitoring., RFC 8321 (2018) 1–33.
[171] T. Pan, E. Song, Z. Bian, X. Lin, X. Peng, J. Zhang, T. Huang, B. Liu, Y. Liu, INT-path: Towards optimal path planning for in-band network-wide telemetry, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 487–495.
[172] N. Katta, M. Hira, C. Kim, A. Sivaraman, J. Rexford, Hula: Scalable load balancing using programmable data planes, in: Proceedings of the Symposium on SDN Research, 2016, pp. 1–12.
[173] C. H. Benet, A. J. Kassler, T. Benson, G. Pongracz, Mp-hula: Multipath transport aware load balancing using programmable data planes, in: Proceedings of the 2018 Morning Workshop on In-Network Computing, 2018, pp. 7–13.
[174] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, L. Vanbever, Blink: Fast connectivity recovery entirely in the data plane, in: 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), 2019, pp. 161–176.
[175] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, et al., Hpcc: High precision congestion control, in: Proceedings of the ACM Special Interest Group on Data Communication, 2019, pp. 44–58.
[176] B. Turkovic, F. Kuipers, N. van Adrichem, K. Langendoen, Fast network congestion detection and avoidance using P4, in: Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies, 2018, pp. 45–51.
[177] E. F. Kfoury, J. Crichigno, E. Bou-Harb, D. Khoury, G. Srivastava, Enabling TCP pacing using programmable data plane switches, in: 2019 42nd International Conference on Telecommunications and Signal Processing (TSP), IEEE, 2019, pp. 273–277.
[178] F. Shaikh, E. Bou-Harb, N. Neshenko, A. P. Wright, N. Ghani, Internet of malicious things: Correlating active and passive measurements for inferring and characterizing internet-scale unsolicited iot devices, IEEE Communications Magazine 56 (9) (2018) 170–177.
[179] Z. Liu, Z. Bai, Z. Liu, X. Li, C. Kim, V. Braverman, X. Jin, I. Stoica, Distcache: Provable load balancing for large-scale storage systems with distributed caching, in: 17th USENIX Conference on File and Storage Technologies (FAST 19), 2019, pp. 143–157.
[180] B. de Haan, Multitenancy and resource management for in-network caching, Ph.D. thesis, Universiteit van Amsterdam (2021).
[181] E. Cidon, S. Choi, S. Katti, N. McKeown, Appswitch: Application-layer load balancing within a software switch, in: Proceedings of the First Asia-Pacific Workshop on Networking, 2017, pp. 64–70.
[182] S. Goldberg, J. Rexford, Security vulnerabilities and solutions for packet sampling, in: 2007 IEEE Sarnoff Symposium, IEEE, 2007, pp. 1–7.
[183] R. Kundel, L. Nobach, J. Blendin, W. Maas, A. Zimber, H.-J. Kolbe, G. Schyguda, V. Gurevich, R. Hark, B. Koldehofe, et al., Openbng: Central office network functions on programmable data plane hardware, International Journal of Network Management 31 (1) (2021) e2134.
[184] R. Shah, V. Kumar, M. Vutukuru, P. Kulkarni, Turboepc: leveraging dataplane programmability to accelerate the mobile packet core, in: Proceedings of the Symposium on SDN Research, 2020, pp. 83–95.
[185] S. K. Singh, C. E. Rothenberg, G. Patra, G. Pongracz, Offloading virtual evolved packet gateway user plane functions to a programmable ASIC, in: Proceedings of the 1st ACM CoNEXT Workshop on Emerging in-Network Computing Paradigms, 2019, pp. 9–14.
[186] P. Palagummi, K. M. Sivalingam, Smartho: A network initiated handover in ng-ran using P4-based switches, in: 2018 14th International Conference on Network and Service Management (CNSM), IEEE, 2018, pp. 338–342.
[187] E. F. Kfoury, J. Crichigno, E. Bou-Harb, Offloading media traffic to programmable data plane switches, in: ICC 2020-2020 IEEE International Conference on Communications (ICC), IEEE, 2020, pp. 1–7.
[188] R. Ricart-Sanchez, P. Malagon, P. Salva-Garcia, E. C. Perez, Q. Wang, J. M. A. Calero, Towards an FPGA-accelerated programmable data path for edge-to-core communications in 5G networks, Journal of Network and Computer Applications 124 (2018) 80–93.
[189] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, N. Zilberman, H. Weatherspoon, M. Canini, F. Pedone, R. Soulé, Partitioned paxos via the network data plane, arXiv preprint arXiv:1901.08806 (2019).
[190] M. Burrows, The chubby lock service for loosely-coupled distributed systems, in: Proceedings of the 7th symposium on Operating systems design and implementation, 2006, pp. 335–350.
[191] Openreplica, [Online]. Available: `https://openreplica.org/`.
[192] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, C. Maltzahn, Ceph: A scalable, high-performance distributed file system, in: Proceedings of the 7th symposium on Operating systems design and implementation, 2006, pp. 307–320.
[193] H. T. Dang, M. Canini, F. Pedone, R. Soulé, Paxos made switch-y, ACM SIGCOMM Computer Communication Review 46 (2) (2016) 18–24.
[194] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, N. Zilberman, H. Weatherspoon, M. Canini, F. Pedone, R. Soulé, P4xos: Consensus as a network service, IEEE/ACM Transactions on Networking 28 (4) (2020) 1726–1738.
[195] J. Li, E. Michael, N. K. Sharma, A. Szekeres, D. R. Ports, Just say NO to paxos overhead: Replacing consensus with network ordering, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 467–483.
[196] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, I. Stoica, Netchain: Scale-free sub-rtt coordination, in: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), 2018, pp. 35–49.
[197] J. Li, E. Michael, D. R. Ports, Eris: Coordination-free consistent transactions using in-network concurrency control, in: Proceedings of the 26th Symposium on Operating Systems Principles, 2017, pp. 104–120.
[198] J. R. Douceur, The sybil attack, in: International workshop on peer-to-peer systems, Springer, 2002, pp. 251–260.
[199] X. Chen, H. Kim, J. M. Aman, W. Chang, M. Lee, J. Rexford, Measuring TCP round-trip time in the data plane, in: Proceedings of the Workshop on Secure Programmable Network Infrastructure, 2020, pp. 35–41.
[200] N. L. Scouarnec, Cuckoo++ hash tables: High-performance hash tables for networking applications, in: Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems, 2018, pp. 41–54.
[201] D. Kim, Y. Zhu, C. Kim, J. Lee, S. Seshan, Generic external memory for switch data planes, in: Proceedings of the 17th ACM Workshop on Hot Topics in Networks, 2018, pp. 1–7.
[202] V. Bruschi, R. B. Basat, Z. Liu, G. Antichi, G. Bianchi, M. Mitzen-

macher, Discovering the heavy hitters with disaggregated sketches, in: Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies, 2020, pp. 536–537.

[203] D. Ding, M. Savi, D. Siracusa, Estimating logarithmic and exponential functions to track network traffic entropy in P4, in: NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2020, pp. 1–9.

[204] N. K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, S. Peter, Evaluating the power of flexible packet processing for network resource allocation, in: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), 2017, pp. 67–82.

[205] E. Bou-Harb, M. Debbabi, C. Assi, A novel cyber security capability: Inferring internet-scale infections by correlating malware and probing activities, Computer Networks 94 (2016) 327–343.

[206] C. Pignataro, P. Quinn, Service function chaining, [Online]. Available: `https://tinyurl.com/hsax6vyt`.

[207] S. Han, S. Jang, H. Choi, H. Lee, S. Pack, Virtualization in programmable data plane: A survey and open challenges, IEEE Open Journal of the Communications Society 1 (2020) 527–534.

[208] D. Hancock, J. Van der Merwe, Hyper4: Using P4 to virtualize the programmable data plane, in: Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies, 2016, pp. 35–49.

[209] C. Zhang, J. Bi, Y. Zhou, J. Wu, Hypervdp: High-performance virtualization of the programmable data plane, IEEE Journal on Selected Areas in Communications 37 (3) (2019) 556–569.

[210] P. Zheng, T. Benson, C. Hu, P4visor: Lightweight virtualization and composition primitives for building and testing modular programs, in: Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies, 2018, pp. 98–111.

[211] R. Stoyanov, N. Zilberman, Mtpsa: Multi-tenant programmable switches, in: Proceedings of the 3rd P4 Workshop in Europe, 2020, pp. 43–48.

[212] D. Wu, A. Chen, T. E. Ng, G. Wang, H. Wang, Accelerated service chaining on a single switch ASIC, in: Proceedings of the 18th ACM Workshop on Hot Topics in Networks, 2019, pp. 141–149.

[213] T. Jepsen, D. Alvarez, N. Foster, C. Kim, J. Lee, M. Moshref, R. Soulé, Fast string searching on pisa, in: Proceedings of the 2019 ACM Symposium on SDN Research, 2019, pp. 21–28.

[214] J. Hypolite, J. Sonchack, S. Hershkop, N. Dautenhahn, A. DeHon, J. M. Smith, Deepmatch: practical deep packet inspection in the data plane using network processors, in: Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies, 2020, pp. 336–350.

[215] Nfp-6000 intelligent ethernet controller family, [Online]. Available: `https://tinyurl.com/r3bkbjwj`.

[216] Y. Shi, M. Wen, C. Zhang, Incremental deployment of programmable switches for sketch-based network measurement, in: 2020 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2020, pp. 1–7.

[217] T. Mai, S. Garg, H. Yao, J. Nie, G. Kaddoum, Z. Xiong, In-network intelligence control: Toward a self-driving networking architecture, IEEE Network 35 (2) (2021) 53–59.