P4Tune: Enabling Programmability in Non-Programmable Networks

Article *in* IEEE Communications Magazine · January 2023

Elie Kfoury
University of South Carolina

Jorge Crichigno
University of South Carolina

Elias Bou-Harb
University of Texas at San Antonio

P4Tune: Enabling Programmability in Non-Programmable Networks

Elie Kfoury*, Jorge Crichigno*, Elias Bou-Harb†
*University of South Carolina, Columbia, South Carolina - USA
†University of Texas at San Antonio, San Antonio, Texas - USA

***Abstract*—Data plane programmability has attracted significant attention, permitting network operators to run customized packet processing functions. Unfortunately, networks today are still largely dominated by proprietary fixed-function devices. It is challenging for operators to completely migrate to programmable data planes (PDPs) due to the economical costs of replacing equipment and limited expertise in maintaining and operating programmable networks. This paper introduces P4Tune, a cost-efficient architecture that uses passive PDPs coupled with optical taps. P4Tune runs customized packet processing functions operating at line rate and collects network information with nanosecond resolution. This visibility enables intelligent algorithms residing in the control plane to construct configuration rules and apply them on the legacy devices, thus creating a closed control loop. P4Tune brings the benefits of processing traffic at line rate to legacy networks. Additionally, by using PDPs passively, the architecture can be safely deployed without disrupting current network operations, fostering incremental use of PDPs (i.e., no need to deploy complex code at once). The paper demonstrates three applications of the proposed architecture related to Quality of Service (QoS) and cybersecurity.**

***Index Terms*—Programmable data planes, closed control loop, incremental deployment, self-driving networks, P4.**

I. Introduction

FOR decades, the networking industry has been operating in a bottom-up approach. The vendors provide devices with fixed-function Application-specific Integrated Circuits (ASICs), which limits the protocols and features available to network operators. Recently, programmable switches have emerged to customize the behavior of the data plane [1]. As a result, a wide range of in-network applications have been developed, including caching, load balancing, congestion control, network telemetry, and others [2]. The literature has shown that offloading applications to the data plane provides significant performance gains and greater visibility into the behavior of the network. However, although Programmable Data Planes (PDPs) have been adopted by top cloud providers such as Google [3] and Facebook [4] -which have large teams of software developers and engineers- most campus and enterprise networks, and smaller service providers are still using legacy (non-programmable data plane) devices.

There are many reasons why programmable switches are not used in most production networks: i) network operators are accustomed to the bottom-up approach where vendors provide non-programmable but configurable devices, allowing the operators to only perform device configuration (e.g., interfaces, routing engine); ii) programmable switches have limited support from vendors, as these devices are white-boxes; iii) replacing the devices in a production network incurs significant costs and may interrupt the services provided by the network; iv) substantial expertise is needed to fully replace a legacy network.

The fixed-function devices available in networks today (e.g., core routers/switches) contain a set of primitives that can influence the traffic crossing the network. Examples of such primitives include packet dropping, rate limiting, bandwidth allocation, and buffer sizing. Such primitives are often configured only statically, limiting their potential in networks with dynamic workloads. Thus, they are often underutilized. Since fixed-function devices are not programmable, they cannot run customized algorithms that leverage these primitives.

This article proposes P4Tune, a framework that brings the benefits of processing traffic at line rate in the PDPs to non-programmable networks. Optical taps are installed on the links of the legacy network devices. The traffic is forwarded to PDPs where in-network applications are running. The measurements and statistics computed by the PDPs are used by intelligent algorithms running on CPUs. These algorithms then construct configuration rules and apply them to legacy devices, creating a closed control loop.

The main contributions of this paper are framed as follows.

1) Designing a cost-efficient framework that leverages the visibility and performance of PDPs in a legacy (non-programmable) network.
2) Enabling network operators to incrementally use passive PDPs, without requiring them to deploy complex code at once. This would avoid disrupting current network operations.
3) Demonstrating the benefits brought by PDPs with three use cases. Two use cases improve the Quality-of-Service (QoS) of the legacy network, and one use case mitigates a common cyberattack.

The rest of the paper is organized as follows. Section II provides a background on PDPs and the related work, and describes the challenges faced in deploying PDPs in contemporary networks. Section III describes the proposed framework and discusses how the layers interact to provide a closed control loop. Sections IV and V demonstrate the use cases of QoS and cybersecurity, respectively. Section VI presents further discussions on P4Tune, its limitations, and the lessons learned. Finally, Section VII summarizes the paper.

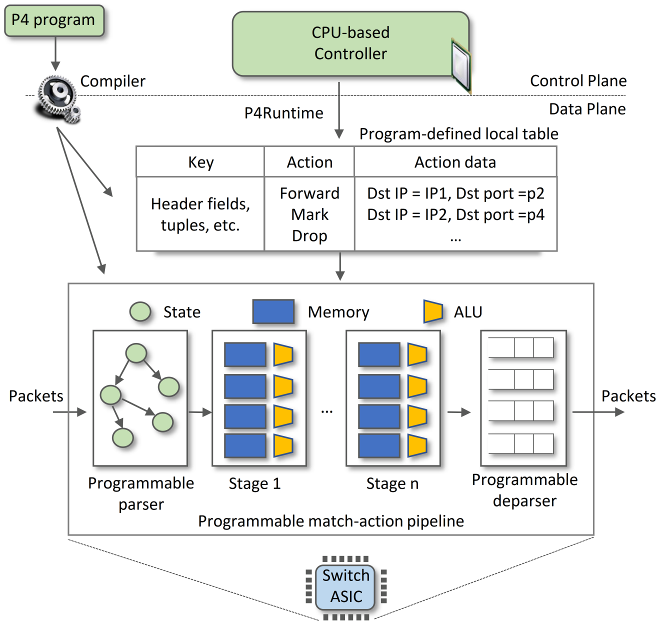

Fig. 1. Programmable data plane and its interaction with the control plane.

II. Background and Related Work

This section provides a general overview of PDPs and discusses the related work and the PDPs' deployment challenges.

A. Brief Background on Programmable Data Planes

Consider Fig. 1 which shows a PDP and its interaction with the control plane. The PDP includes the following elements:

- Programmable parser: allows the programmer to define the headers according to custom or standard protocols, and to parse them. The parser is represented as a state machine.
- Programmable match-action pipeline: executes operations over the packet headers and intermediate results. A single match-action stage has multiple memory blocks and Arithmetic Logic Units (ALUs). The memory blocks are used to implement tables containing keys to match against header fields or tuples, and the ALUs are used to execute actions. Since some action results may be needed for further processing, stages are arranged sequentially.
- Programmable deparser: assembles the packet headers back and serializes them for transmission.

The ALUs execute simple operations and use a Reduced Instruction Set Computer (RISC)-type instruction set, therefore they can be implemented at a minimal cost and size.

The performance gain of a PDP relies on the multiple dimensions of parallelism. With tens of stages per PDP, hundreds of match-action units per stage, and thousands of ALUs per chip, the packet processing speed of a PDP can be in the order of tens of terabits per second (Tbps), a few orders of magnitude faster than that of a general-purpose CPU.

The *de facto* language for describing the PDP behavior is P4 [1]. P4 is a domain-specific language for networking that stands for Programming Protocol-independent Packet Processors. The P4 compiler accepts as input a P4 program and generates a binary code that is loaded into the PDP. It also generates the APIs used by the control plane to populate table entries with keys and action data. P4Runtime is the specification that describes the protocols used to manipulate table entries at run time.

In the past few years, a wide range of P4-based in-network applications have been developed, including caching, load balancing, congestion control, telemetry, cyberattacks defenses, and others [2].

B. Related Work

P4Campus [5], a recent project managed by researchers from Princeton, provides a guideline for P4 researchers to test their ideas on campus networks. It encourages engineers to use optical taps to copy packets and forward them to programmable switches. P4Campus focuses on the challenges of convincing operators to deploy P4 switches. P4Campus is limited to entirely passive applications (e.g., monitoring the queue latency, measuring the round-trip time (RTT), fingerprinting the operating systems) which do not impact real-time network traffic.

Deep programmability [6] refers to an ambitious vision where elements in the network, vertically (control and data planes) and horizontally (end-to-end), are fully programmable. Telemetry data (i.e., “dials”), which are collected from PDPs, are used to observe the packets and the network state. Knobs, on the other hand, are used to control the network, creating a verifiable closed loop. This framework gives control to network owners rather than to equipment vendors and technology providers. While promising, this vision is yet to be realized due to various reasons discussed in Section II-C.

C. PDPs Deployment Challenges

1) Data Plane Programmability Knowledge by Operators: Network operators are accustomed to the bottom-up approach where fixed-function devices are provided by vendors. These devices include standard packet processing features (e.g., switching, routing, access control lists (ACLs)) that operate on standardized protocols. Operators can only configure legacy devices (e.g., modify routing configuration, update ACLs) rather than implementing their own data plane functions. Adopting PDPs would require a deep understanding of each component of the forwarding pipeline (parser, match-action tables, etc.). Moreover, as the number of features increases, the complexity of the P4 program greatly increases. Thus, deploying a fully programmable network may require a team of software developers and network engineers with substantial expertise.

2) Cost of Replacing the Existing Infrastructure : A significant amount of time and money has been spent to build existing network infrastructures. They consist of routers, switches, security appliances, and other middlebox devices. Replacing the current network infrastructure would require topology modifications, new configurations, the implementation of legacy and customized data plane functions, along with new expertise to carry on the above tasks. Thus, from the logistic and cost perspectives, migrating entirely to a programmable infrastructure may not be a viable option.

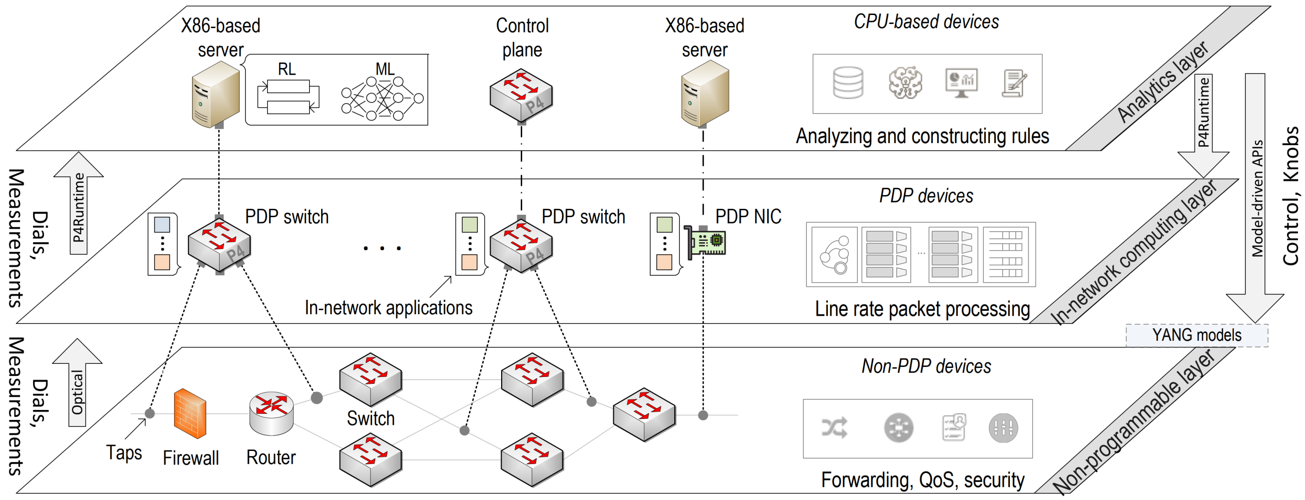

Fig. 2. Generic framework.

3) Vendor Support, Maintenance, and Network Availability: In contrast to legacy devices where support from vendors is readily available, PDPs are mostly maintained and supported by an in-house team of developers and engineers. Without appropriate development and support, sources of packet processing errors may increase and be difficult to troubleshoot, leading to network disruptions and service unavailability. Such risk is typically not tolerated in production networks [6].

III. Proposed Framework

P4Tune builds on ideas from both P4Campus and deep programmability to provide a cost-efficient solution that can be easily deployed in today's networks. Unlike P4Campus which is deployed entirely passively, P4Tune is able to control the network and close the feedback loop. Unlike deep programmability which assumes that the entire network is programmable, P4Tune is able to reconfigure legacy devices deployed in today's networks.

A. Overview

Consider Fig. 2 which shows an overview of P4Tune. The goal of this framework is to leverage the power of PDPs to enhance performance in a non-programmable network. This is achieved by applying custom algorithms at line rate in the data plane over traffic copied from the legacy devices. These in-network algorithms produce statistics that can be used by intelligent algorithms in the control plane to construct configuration rules and policies. Finally, policies are applied to the legacy devices.

B. Layers

1) Non-programmable Layer: This layer consists of legacy (i.e., non-programmable) devices that provide traditional functionalities such as forwarding, QoS, and access control. Such devices include routers, switches, and middleboxes (e.g., firewalls, load balancers). They are configurable (e.g., configuring basic security policies through an ACL) but not programmable. Modern legacy devices may also implement a wide range of features beyond basic forwarding, including traffic classification, policing, queuing, and scheduling. The configuration is typically static and performed by an operator through a command-line interface. Thus, such hard-coded static solution is far from the optimal, as traffic patterns vary over time.

2) In-network Computing Layer: Traffic crossing network devices can add up to hundreds of gigabits per second. Real-time packet processing at such rates is not possible on general-purpose CPUs. This layer consists of PDPs with processing speeds in the order of terabits per second. They run in-network applications with customized packet processing algorithms. PDPs not only provide high-precision timers (nanosecond granularity [2]) but also stateful memories (e.g., registers, counters, meters) that can be accessed at line rate. Such capabilities enable a wide range of in-network applications such as identifying heavy hitters, detecting Denial of Service (DoS) attacks, measuring microbursts, etc.

Devices in this layer operate over a copy of the traffic provided through the passive optical taps. They are capable of processing packets in real-time, thus providing high visibility and accurate measurements to the upper layer.

3) Analytics Layer: This layer consists of general-purpose CPUs residing on x86-based servers or on the switch's control plane, and potentially hardware accelerators such as graphic processing units (GPUs) and tensor processing units (TPUs). Devices can analyze the information received from the In-network Computing layer and execute advanced algorithms, such as Machine Learning (ML) and Reinforcement Learning (RL) algorithms, to make control decisions. Other algorithms may include passive monitoring and visualization via dashboards, to assist administrators with the management of the network.

C. Interaction between Layers

Traffic is duplicated through optical taps from the Non-programmable layer to the In-network Computing layer. Taps preserve timing information and create an exact copy of all

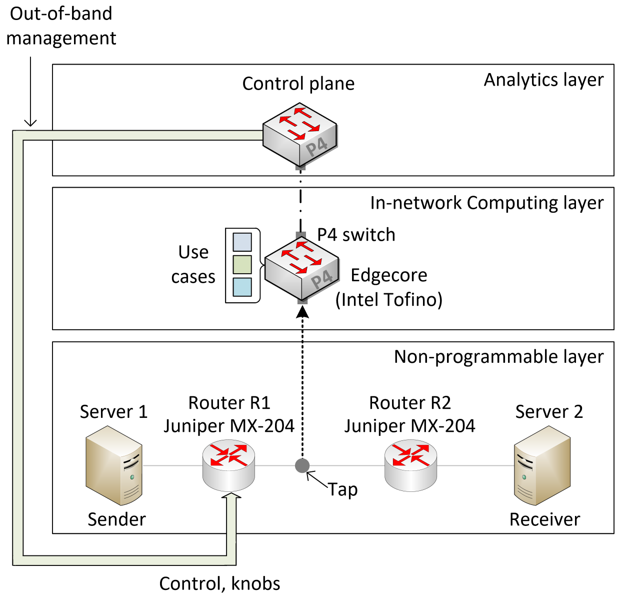

Fig. 3. Topology used for the experiments.

packets, in contrast to port mirroring which can incur packet reordering, drops, and incorrect arrival times. Furthermore, taps do not consume an additional port on the device and are cheap and easy to deploy. Not all links require taps, as the network operator can specify the vantage points, depending on the application of interest. Once the traffic is received by the PDPs in the In-network Computing layer, customized algorithms are executed. They compute statistics and measurements that can be leveraged by the Analytics layer. The connection between these two layers can be via physical links (e.g., via a fiber optic or other medium) or via virtual links (e.g., connecting the data plane to the control plane via a virtual Ethernet link). The Analytics layer receives the statistics and measurements produced by the devices in the In-network Computing layer and runs algorithms to generate configuration rules that are then applied to the legacy devices in the Non-programmable layer. The Analytics layer uses model-driven APIs and YANG models. Essentially, YANG models (OpenConfig, IETF, native) residing on the non-PDP devices will define the device's OS configuration hierarchy and operational commands. Applications running on the Analytics layer will use APIs to interact with the models to configure the legacy devices. To make the solution hardware-agnostic, OpenConfig models are used. OpenConfig is a collaborative effort by network operators to develop programmatic interfaces for managing networks in a vendor-neutral way. OpenConfig is supported on devices from major providers including Cisco, Juniper, and Arista.

The Analytics layer can also reconfigure devices in the In-network Computing layer (e.g., modify a threshold) through P4Runtime. The P4Runtime API is a control plane specification for controlling the PDP elements of a device. P4Runtime is open-source and silicon-independent. Note that the In-network Computing layer does not configure legacy devices in the Non-programmable layer.

The continuous interaction between the layers creates a closed control loop, enabling an autonomous, self-driving network, even in the presence of legacy devices.

D. Experimental Setup

The use cases in this paper are implemented over the topology shown in Fig. 3. The topology consists of the following devices:

- Non-programmable layer: Juniper MX-204 is the legacy router used. This router supports traffic classification and allows buffer size reconfiguration. Traffic is generated and received on general-purpose servers with enough computing and memory resources.
- In-network Computing layer: Edgecore Wedge100BF-32X is the PDP used. It is equipped with a programmable ASIC chip (Intel Tofino [7]) that operates at 3.2 Tbps.
- Analytics layer: the control plane of the Edgecore switch is used.

The control plane in the Analytics layer reconfigures the Juniper router R1 in the Non-programmable layer through an out-of-band management connection.

IV. Use Case 1: Quality of Service

Packet-switched devices (*e.g.*, routers, switches) are equipped with buffers that accommodate transient bursts and reduce packet drops. Buffer management (configuring its size, separating its crossing flows) significantly impacts the performance of network applications.

A. Dynamic Buffer Sizing

The rule-of-thumb buffer size that ensures high link utilization is $B = C \cdot RTT_{min}$, where C is the link capacity and RTT_{min} is the average minimum RTT. [8] showed that much smaller buffers can be used without having a significant decrease in the link utilization (98%). This rule suggests that $B = \frac{C \cdot RTT_{min}}{\sqrt{N}}$, where N is the number of large flows that are persistent over time.

According to [9], most routers in production networks use very large *-bloated-* buffers, increasing RTT to a point where the latency of small flows is heavily affected during congested periods. Specifying a static buffer size does not yield to optimal performance since the optimal buffer size is a function of the average RTT and the number of large flows, which change over time. The idea of dynamic buffer sizing was never implemented because of the lack of visibility and high-resolution measurements to i) discriminate between large and small flows, ii) compute the number of flows passing through the device in real-time, and iii) calculate the RTT of thousands of concurrent flows at any given time. This information would require computation at line rate.

In this use case, a control plane's CPU in the Analytics layer dynamically modifies a legacy router's buffer size by using measurements obtained from a PDP in the In-network Computing layer. The goal is to achieve small Flow Completion Times (FCT) for small flows traversing the router during congested periods.

TABLE I
FLOW COMPLETION TIMES AND ROUND-TRIP TIMES OF SMALL FLOWS

	wo/ buffer modification				w/ buffer modification			
CCA	FCT [s]		RTT [ms]		FCT [s]		RTT [ms]	
	μ	σ	μ	σ	μ	σ	μ	σ
Cubic	1.84	0.84	207.8	18.5	0.45	0.51	26.9	2.3
Vegas	0.82	0.61	82.9	34.7	0.47	0.46	27.7	2.5
Reno	1.58	0.58	193.6	24.5	0.46	0.47	27.0	2.3
BBRv2	0.80	0.68	75.7	32.7	0.57	0.16	29.5	2.7
HTCP	1.62	0.75	186.2	21.2	0.44	0.44	27.8	2.3
DCTCP	1.62	0.70	195.6	24.3	0.45	0.45	27.9	2.6

1) Metrics Estimation: Setting the value of the buffer size to $\frac{C\cdot RTT_{min}}{\sqrt{N}}$ requires determining the current RTT_{min} and N.

The PDP can identify large flows by using the Count-Min Sketch (CMS) data structure. The CMS tracks the number of packets for a given flow. If this number exceeds a threshold, the flow is considered large, and N is incremented by one. As for the RTT calculation, the PDP relates the sequence and acknowledgment numbers of TCP to identify the two directions of the stream and then computes the time difference to produce an RTT sample [10].

Having both, N and the average RTT, the CPU computes the buffer size and modifies the existing buffer size on the legacy device.

2) Evaluation and Results: To evaluate this use case, 100 large flows are generated over a bottleneck link of 1Gbps with 25ms of propagation delay. In addition, the experiment initiated $10,000$ small flows whose inter-connection times are generated from an exponential distribution with a mean of one second. The goal of this test is to measure the FCTs and the RTTs of small flows. The results of this use case are compared against those obtained with the bloated buffer since the latter is the most commonly configured buffer size on the Internet [9]. The experiment also considers flows belonging to different Congestion Control Algorithms (CCAs) such as Cubic and BBRv2.

Table I shows the mean (μ) and the standard deviation (σ) of flows' FCTs and RTTs. Independently of the CCA, dynamically modifying the buffer size (denoted as w/ buffer modification in the table) always yields an improvement in the FCT and the RTT of small flows. As for the large flows, the average FCT only increases by ≈ 0.5% with the dynamic buffer approach. Such an increase is tolerable as large flows are generally non-interactive.

Additionally, Jain's fairness index, which quantifies how fairly the link is shared by flows, increases from $\approx 59.5\%$ with the static buffer approach to $\approx 82.3\%$.

B. Size-aware Flow Separation

The FCT of small flows sharing a router queue with large flows is significantly impacted when the network is busy. Web traffic can be considered a workload that contains small flows. A possible solution to prevent the increase of FCTs is to separate small flows from large flows. This can be achieved by classifying traffic in core routers and separating them into different queues. There are two main typical classifiers available in commercial routers: 1) Behavior Aggregate Classifiers: a classifier that leverages the fixed-length fields in the packet header (e.g., IP precedence, Differentiated Services Code Point (DSCP)); 2) Multifield Classifier (MF): a classifier that examines multiple fields in the packet (e.g., source/destination addresses/port, TCP flags, protocol, packet length) based on firewall filter rules.

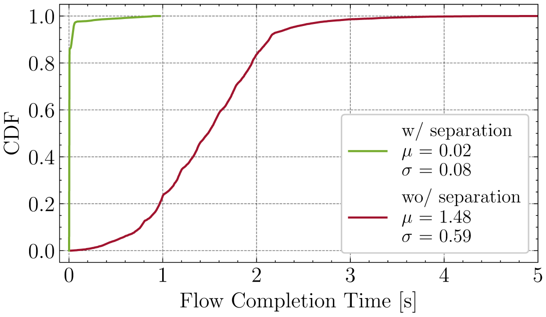

Fig. 4. Cumulative Distribution Function of the FCT of small flows.

The idea of this use case is to separate large and small flows passing through a legacy router into different queues. A PDP in the In-network Computing layer identifies large flows. After receiving a notification from the PDP, a control plane's CPU in the Analytics layer allocates dedicated queues for large and small flows in the legacy router to store their packets accordingly. This use case uses the MF classifier for the classification on the legacy router.

1) Evaluation and Results: To evaluate this use case, 10,000 small flows are generated (same as in Section IV-A2). In addition, 10 large flows were started, each with a random starting time over the test duration. The test is executed with the default settings of the router (wo/ separation), and with the proposed approach (w/ separation).

Fig. 4 shows the FCT of small flows. It can be seen that there is a significant improvement in the FCT of the small flows; more than 90% of the flows finished in less than 0.2 seconds. On the other hand, $\approx 75\%$ of the non-separated flows terminated in more than one second.

V. USE CASE 2: CYBERSECURITY

The diversity and the scale of attacks have dramatically increased in the past few years. In January 2022, a record-breaking attack reached a throughput of 3.47 terabits per second [11]; such rates cannot be promptly detected and mitigated with contemporary defense solutions, neither software-based nor expensive and proprietary devices.

Recent efforts have shown that a wide range of defenses can be devised in programmable switches.

A. DNS Amplification

DNS amplification is an attack where a massive amount of DNS response packets is sent to a victim's server. The attacker sends a DNS name lookup request to a public DNS resolver using the victim's IP as the source address, causing the DNS response to be sent to the spoofed address. Typically,

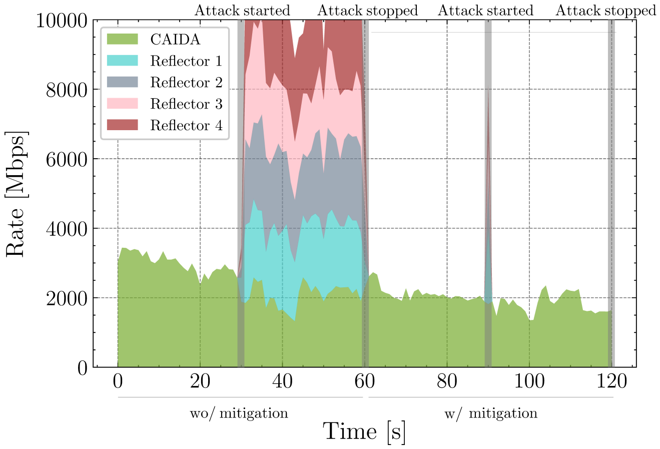

Fig. 5. Amplification attack results. The attack is augmented to the CAIDA traffic at intervals [30-60] and [90-120]. Note how the attack was promptly stopped at second 90.

an attacker gathers as much zone information as possible to maximize the amplification effect.

The implementation of this use case consists of programming a PDP to parse DNS packets. The data plane has visibility of all DNS requests and responses that pass through a legacy router, which will be protected. The PDP tracks the number of DNS responses that do not match the DNS requests sent from the network to the public resolvers. The counts are tracked per resolver, allowing the system to fingerprint the IP address of any resolver being attacked. The PDP device notifies a control plane's CPU in the Analytics layer of the volume of DNS requests and responses. Once an attack is detected, the CPU constructs a rule to be applied to the legacy router. The MF classifier matches on the IP address of the attacked resolver and on the lengths of the packets sent from that resolver. Upon matching, the legacy router discards the packets. Using the packet length as an additional match criterion allows legitimate small-sized DNS responses received from the resolver to be forwarded by the router.

1) Evaluation and Results: The goal of this test is to determine whether the system can detect the amplification, and the time it takes for the attack to be mitigated. While replaying a CAIDA dataset, which was captured from high-speed monitors on a commercial backbone link [12], a DNS amplification attack is launched. Fig. 5 shows the receiving rate on the victim, with and without mitigation. During the first 30 seconds, only legitimate CAIDA traffic was being sent; a second later, an amplification attack was launched. It can be seen that without mitigation, the reflectors flooded the 10G link rapidly. With the mitigation activated, the amplification attack was promptly detected and stopped (less than a second), as seen in second 90 in the figure. It is worth noting that the configuration commit time is constant, regardless of the traffic rate (i.e., even if this attack was in the Tbps scale).

VI. Discussions

This section presents some discussions on the proposed framework.

A. Cost-efficiency

P4Tune requires installing PDP switches and optical passive taps on the links to be monitored. PDP switches are cheaper than most legacy switches used on campus/enterprise networks. Taps are also cheap and can be installed without causing any changes to the existing topology (e.g., IP addressing, routing). However, note that installing taps would necessitate disconnecting cables, leading to a short disruption of the network during their installation.

B. Promptness

While P4Tune does not apply the configuration rule at line rate (the Analytics layer, e.g., a control plane's CPU, applies the rules to a legacy device), the PDP switches still perform packet processing at line rate. This capability enables the system to detect events (e.g., flood attacks, amplification cases) and to measure parameters (e.g., RTT, number of active flows) with high precision. The control loop is closed by committing the configuration rule on the legacy device. According to the experimental results, the commit latency is less than one second, regardless of the number of rules installed with the single commit.

C. Supported Primitives

The applications that control the network are limited to the primitives provided by the fixed-function devices. Common primitives found in commercial devices include dropping packets, modifying the bandwidth allocated to a queue, changing queue size and loss priorities, metering and policing traffic, modifying routing tables, etc. These primitives, if used in conjunction with PDPs, can enable a plethora of applications with dynamic workloads, including traffic rerouting, load balancing, traffic steering, botnet prevention, etc.

D. Unsupported Applications

The current framework does not support active applications running on PDPs that send feedback to the end devices (e.g., HPCC [13], a congestion control scheme that sends congestion feedback to the senders). Instead, P4Tune relies on devices in the Analytics layer (e.g., the control plane's CPUs) to close the control loop. However, P4Tune can be extended to provide feedback from the In-network Computing layer by generating control signals directly from the PDPs.

E. Privacy

There is always concern about protecting user privacy in network traffic. Fortunately, there is a way to use PDPs without violating privacy. P4Campus presents an online network traffic anonymization application [14] that is written in P4 and is compatible with P4Tune.

F. Lessons Learned

1) Lesson 1 - Visibility and Incremental Adoption of PDPs: While PDPs were initially created to permit programmers to develop new protocols, they also provide real-time visibility and processing speed previously unavailable. These features allowed the authors to accurately measure RTTs and identify long flows and unsolicited DNS responses. Additionally, by using passive PDPs and operating over a copy of the traffic, the system becomes less disruptive and fosters incremental use of PDPs (i.e., no need to deploy complex code at once).

2) Lesson 2 - PDPs Constraints: PDPs have limited computing and memory resources. It is essential to identify an optimized way to implement the application on a PDP. For instance, counting packets pertaining to a flow can be implemented by maintaining counts in a single register array, indexed by the flow identifier. However, this design is not scalable. Instead, by using a memory-efficient data structure such as the CMS, many more flows can be tracked.

3) Lesson 3 - Open APIs for Managing Legacy Devices: Networks nowadays encompass legacy devices from different vendors (e.g., Cisco, Juniper, Arista). Recent efforts in the networking community have been devoted on developing common vendor-independent software layers for managing network devices. The proposed P4Tune scheme indeed exploited such vendor-independent APIs, which enable seamless portability of applications.

4) Lesson 4 - Primitives Underutilization in Legacy Devices: Legacy devices support a wide range of primitives that can be leveraged to control the network. Most of these primitives require traffic to be classified. Classification can be achieved by inspecting header fields or by matching on pre-configured flow entries. Recent studies have shown that the fields in the IP header related to traffic differentiation are rarely used in traffic crossing a well-known backbone network [15]. On the other hand, pre-configuring the flows require manual intervention from the operator, and thus is typically limited to well-known subnets. By leveraging passive PDPs (as proposed in this paper), traffic can be classified using custom algorithms running at line rate. Subsequently, the Analytics layer can install flow entries in the legacy devices for traffic classification. Such architecture enables the primitives on the legacy devices to be further utilized.

VII. Conclusion

This paper presented P4Tune, a cost-efficient architecture that uses passive PDPs to run custom packet processing on the traffic traversing the legacy network. Intelligent algorithms residing on the general-purpose CPUs construct and push configuration rules to the legacy devices, creating a closed control loop. The article demonstrated three different applications leveraging the proposed framework, namely, dynamic router buffer sizing, traffic separation, and DNS amplification mitigation. In this article, P4Tune was used with the implementation of just several interesting use cases as examples. However, due to the flexibility and speed of PDPs, many more applications that ameliorate the network can be developed.

Acknowledgments

This work was supported by the U.S National Science Foundation (NSF), Office of Advanced Cyberinfrastructure, Awards #1925484 and #2118311.

References

[1] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[2] E. F. Kfoury *et al.*, "An exhaustive survey on P4 programmable data plane switches: taxonomy, applications, challenges, and future trends," *IEEE Access*, vol. 9, pp. 87094–87155, 2021.

[3] S. Heule, "Using P4 and P4Runtime for optimal L3 routing." [Online]. Available: https://tinyurl.com/y365gnqy, Accessed on October 14, 2022.

[4] Facebook engineering, "Disaggregate: networking recap." [Online]. Available: https://tinyurl.com/yxoaj7kw, Accessed on October 14, 2022.

[5] H. Kim *et al.*, "Experience-driven research on programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 51, no. 1, pp. 10–17, 2021.

[6] N. Foster *et al.*, "Using deep programmability to put network owners in control," *ACM SIGCOMM Computer Communication Review*, vol. 50, no. 4, pp. 82–88, 2020.

[7] Intel, "Intel Tofino, P4-programmable Ethernet switch ASIC that delivers better performance at lower power." [Online]. Available: https://tinyurl.com/2vzssub3, Accessed on December 15, 2022.

[8] G. Appenzeller *et al.*, "Sizing router buffers," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 281–292, 2004.

[9] N. McKeown *et al.*, "Sizing router buffers (redux)," *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 5, pp. 69–74, 2019.

[10] X. Chen *et al.*, "Measuring TCP round-trip time in the data plane," in *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, pp. 35–41, 2020.

[11] Microsoft, "Azure DDoS Protection—2021 Q3 and Q4 DDoS attack trends." [Online]. Available: https://tinyurl.com/3zxt3mun, Accessed on October 14, 2022.

[12] CAIDA, "The CAIDA anonymized Internet traces 2019 dataset." [Online]. Available: https://data.caida.org/datasets/passive-2019/, Accessed on October 14, 2022.

[13] Y. Li *et al.*, "Hpcc: High precision congestion control," in *Proceedings of the ACM Special Interest Group on Data Communication*, pp. 44–58, 2019.

[14] H. Kim and A. Gupta, "Ontas: Flexible and scalable online network traffic anonymization system," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, pp. 15–21, 2019.

[15] N. Roddav *et al.*, "On the usage of DSCP and ECN codepoints in Internet backbone traffic traces for IPv4 and IPv6," in *2019 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–6, IEEE, 2019.

Elie Kfoury is pursuing a Ph.D. degree at the College of Engineering and Computing, University of South Carolina (USC), USA.

Jorge Crichigno is a Professor in the College of Engineering and Computing at the University of South Carolina (USC) and the director of the Cyberinfrastructure Lab at USC. He has over 20 years of experience in the academic and industry sectors. Dr. Crichigno's research focuses on offloading functionality to P4 PDP switches, network security, and IoT devices. His work has been funded by private industry and U.S. agencies such as the National Science Foundation (NSF), the Department of Energy (DOE), and the Office of Naval Research (ONR). He received his Ph.D. in Computer Engineering from the University of New Mexico, USA, in 2009.

Elias Bou-Harb is the Director of the Cyber Center For Security and Analytics at UTSA. He is also a tenured Associate Professor at the department of Information Systems and Cyber Security specializing in operational cybersecurity and data science as applicable to national security challenges.