An Exhaustive Survey on P4 Programmable Data Plane Switches: Taxonomy, Applications, Challenges, and Future Trends

Elie F. Kfoury*, Jorge Crichigno*, Elias Bou-Harb†
*College of Engineering and Computing, University of South Carolina, Columbia, U.S.A.
†The Cyber Center For Security and Analytics, University of Texas at San Antonio, U.S.A.

***Abstract*—Traditionally, the data plane has been designed with fixed functions to forward packets using a small set of protocols. This closed-design paradigm has limited the capability of the switches to proprietary implementations which are hard-coded by vendors, inducing a lengthy, costly, and inflexible process. Recently, data plane programmability has attracted significant attention from both the research community and the industry, permitting operators and programmers in general to run customized packet processing functions. This open-design paradigm is paving the way for an unprecedented wave of innovation and experimentation by reducing the time of designing, testing, and adopting new protocols; enabling a customized, top-down approach to develop network applications; providing granular visibility of packet events defined by the programmer; reducing complexity and enhancing resource utilization of the programmable switches; and drastically improving the performance of applications that are offloaded to the data plane.**

Despite the impressive advantages of programmable data plane switches and their importance in modern networks, the literature has been missing a comprehensive guideline in the form of tutorial and survey. To this end, this paper addresses this gap by providing a tutorial encompassing an overview of the evolution of networks from legacy to programmable; describing the essentials of programmable switches and the de-facto programming language, P4; and summarizing the advantages of programmable switches over SDN and legacy devices. The paper further contributes by presenting a unique, comprehensive taxonomy of applications developed with P4; surveying, classifying, and analyzing more than 130 articles; discussing challenges and considerations; and presenting future perspectives and open research issues.

***Index Terms*—Programmable switches, P4 language, Software-defined Networking, data plane, custom packet processing, taxonomy.**

I. Introduction

Since the emergence of the world wide web and the explosive growth of the Internet in the 1990s, the networking industry has been dominated by closed and proprietary hardware and software. Consider the observations made by McKeown [1] and the illustration in Fig. 1, which shows the cumulative number of Request For Comments (RFCs) that describe protocols, processes, middle-box functions, and other network-related standards [2]. While at first an increase in RFCs may appear encouraging, it has actually represented an entry barrier to the network market. The progressive reduction in the flexibility of protocol design caused by standardized requirements,

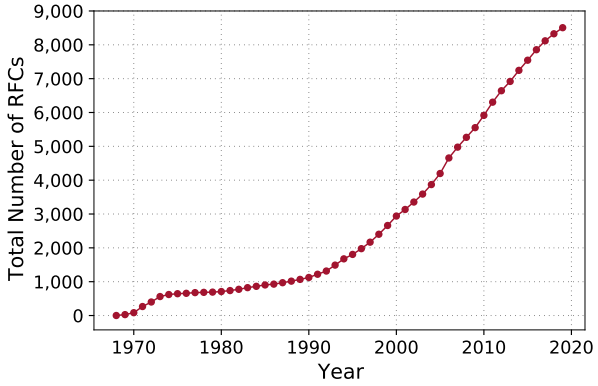


Fig. 1. Cumulative number of RFCs.

which cannot easily be removed to enable protocol changes, has perpetuated the status quo. This protocol ossification [3, 4] has been characterized by a slow innovation pace at the hand of few network vendors. As an example, after initially conceived by Cisco and VMware [5], the Application Specific Integrated Circuit (ASIC) implementation of the Virtual Extensible LAN (VXLAN) [6], a simple frame encapsulation protocol, took several years, a process that could have been reduced to weeks by software implementations[1].

Protocol ossification has been challenged first by Software-defined Networking (SDN) [7, 8] and then by the recent advent of programmable switches. SDN fostered major advances by explicitly separating the control and data planes, and by implementing the control plane intelligence as a software outside of the switches. While SDN reduced network complexity and spurred control plane innovation at the speed of software development, it did not wrest control of the actual packet processing functions away from network vendors. Traditionally, the data plane has been designed with fixed functions to forward packets using a small set of protocols (e.g., IP, Ethernet). The design cycle of switch ASICs has been characterized by a lengthy, closed, and proprietary process that usually takes years. Such process contrasts with the agility of the software industry.

[1]The RFC and VXLAN observations are extracted from Dr. McKeown's presentation during ONF Connect 2019 [1].

The programmable forwarding can be viewed as a natural evolution of SDN, where the software that describes the behavior of how packets are processed can be conceived, tested, and deployed in a much shorter time span. Furthermore, the forwarding behavior can be defined by operators, engineers, researchers, and practitioners in general. The *de-facto* standard for defining forwarding behavior is the P4 language [9], which stands for Programming Protocol-independent Packet Processors. Essentially, P4 programmable switches have removed the entry barrier to network design, previously reserved to network vendors.

The momentum of programmable switches is reflected in the global ecosystem around P4. Operators such as ATT [10], Comcast [11], NTT [12], KPN [13], Turk Telekom [14], Deutsche Telekom [15], China Unicom [14], are now using P4-based platforms and applications to optimize their networks. Companies with large data centers such as Facebook [16], Ali Baba [17], and Google [18] operate on programmable platforms running customized software, a contrast from the fully proprietary implementations of just a few years ago [19]. Switch manufacturers such Edgecore [20], Stordis [21], Cisco [22], Arista [23], Juniper [24], and Interface Masters [25] are now manufacturing P4 programmable switches with multiple deployment models, from fully programmable or white boxes to hybrid schemes. Chip manufactures such as Barefoot Networks (Intel) [26], Xilinx [27], Pensando [28], Mellanox [29], and Innovium [30] have embraced programmable data planes without compromising performance. The availability of tools and agility of software development have opened an unprecedented possibility of experimentation and innovation by enabling network owners to build custom protocols and process them using protocol-independent primitives, reprogram the data plane in the field, and run P4 code on diverse platforms. The main agencies supporting engineering research and education world-wide are investing in programmable networks as well. For example, the U.S. National Science Foundation (NSF) [31] has funded FABRIC [32, 33], a national research backbone based on P4 programmable switches. Another project funded by the NSF operates an international Software Defined Exchange (SDX) which includes a P4 testbed that enables international research and education institutions to share P4 resources [34]. Similarly, an European consortium has recently built 2STiC [35], a P4 programmable network that interconnects universities and research centers.

A. Contribution

Fig. 2 shows the number of papers that cited the original P4 [9] paper versus those that cited the OpenFlow [36] paper, per year. It can be seen that the number of P4-related papers has been increasing since the publication of the original paper in 2014. On the other hand, the number of OpenFlow papers has been decreasing since 2016. This trend can be confirmed through the recent Association for Computing Machinery (ACM) Special Interest Group on Data Communication (SIGCOMM) conference [37]. SIGCOMM is a top-tier conference that accepts computer and data communication networks papers with an average acceptance rate of 8-16%. In SIGCOMM 2020, approximately 30% of the papers are related to data plane programmability and P4; OpenFlow however, is not mentioned in any of the papers.

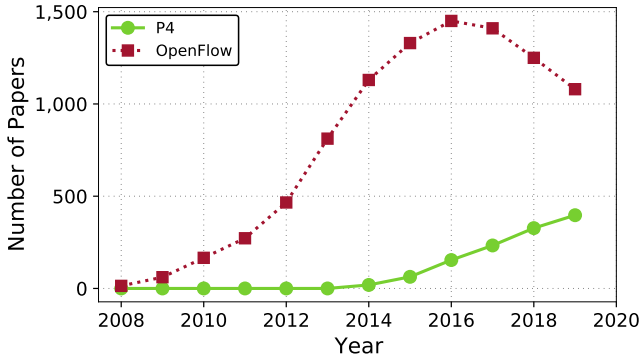


Fig. 2. Number of papers that cited the original P4 [9] and OpenFlow [36] papers per year.

Despite the increasing interest on P4 switches by the industry and academia, previous work has only partially covered this technology. As shown in Table 1, currently, there is no updated and comprehensive material in the forms of tutorials or surveys on P4 switches. Thus, this paper addresses this gap by providing an overview of the evolution of networks from legacy to programmable; describing the essentials of programmable switches, P4 programs, and corresponding workflow; and summarizing the advantages of programmable switches over SDN and legacy devices. The paper continues by presenting a taxonomy of applications developed with P4; surveying, classifying, and analyzing more than 130 articles; discussing challenges and considerations; and putting-forward future perspectives and open research issues.

B. Packet Switches

According to the classical definition, a switch is a layer-2 device that makes forwarding decisions based on the physical address (e.g., Medium Access Control (MAC) address), connecting devices in a local area network. Similarly, a router is a layer-3 device that routes packets across networks.

The distinction between routers and switches has blurred with the introduction of routing functionality into switches and the proliferation of middle-boxes that perform routing, switching, and other operations such as packet filtering and Network Address Translation (NAT). SDN has delivered a unified approach towards providing many of these functions in an integrated manner. A device with such integrated capability is described as a packet switch rather than a layer-3 router or a layer-2 switch.

In-line with the SDN literature, this paper adopts the term *switch* to refer to a general packet switch with integrated layer-2 and layer-3 functionalities.

C. Paper Organization

The road-map of this survey is as follows. Section II studies and compares existing surveys on various P4-related topics and demonstrates the added value of the offered work. Section III

describes the traditional control plane, SDN, and the motivation for programmable data planes. Section IV introduces programmable switches and their features and explains the PISA, a pipeline forwarding model. Section V discusses the building blocks of the P4 language, demonstrates a simple P4 program that achieves IP forwarding, PSA model, P4 and fixed-function devices, Network Operating System (NOS). Section VI describes the workflow of a P4 design that include the compilation process, runtime configuration, and unit testing. The aforementioned sections belong to the tutorial part of this paper. Upcoming sections review the literature pertaining to P4 and data plane programmability, and thus belong to the survey portion of the paper. Section VII describes the survey methodology and the proposed taxonomy. Subsequent sections (from VIII to XIV) explore the works pertaining to various categories proposed in the taxonomy, and compares the P4 approaches in each category to the legacy-enabled solutions. Section XVI outlines challenges and considerations extracted and induced from the literature. Section XVII pinpoints directions that can be explored in the future to ameliorate the state of the art solutions. Finally, Section XVIII concludes the survey. The abbreviations used in this article are summarized in Table XVIII, at the end of the article.

II. Related Surveys

The rapid growth of P4 applications and the benefits that programmable switches present in the networking industry have grabbed considerable attention from the research community. To highlight the latest findings and the research directions, a number of surveys studied the trends and challenges pertaining to programmable switches.

Stubbe et al. [38] discussed various P4 compilers and interpreters in a short survey. This work provided a background on the P4 language and demonstrated the main building blocks that describe packet processing in a programmable switch. This survey outlined the reference implementation of the compiler and the interpreter (i.e., p4c [45] and the Behavioral Model (BMv2) [46]), the Field-Programmable Gate Array (FPGA)-based target (P4FPGA) [47], software switch (PISCES [48]), and proprietary solutions (e.g., Barefoot Capilano [49]).

Dargahi et al. [39] focused on stateful data planes and the security implications that data plane programmability brings. There are two main objectives of this survey. First, it introduces the reader to recent trends and technologies pertaining to stateful data planes. Second, it discusses relevant security issues by analyzing selected use case applications. The survey starts with an introduction on SDN and the rise of stateful data planes. Afterwards, it describes the platforms and enabling technologies that allow programming the data plane. The scope of the survey is not limited to P4 for programming the data plane. Instead, it describes other schemes such as OpenState [50], Flow-level State Transitions (FAST) [51], etc. When reviewing the security properties of stateful data planes, the authors described a mapping between potential attacks and corresponding vulnerabilities. For example, a saturation attack on the switch memory exploits the unbounded flow state memory allocation. Such attack leads to a Denial of Service (DoS) on the switch. The survey was concluded with discussions that elaborate on future research directions.

Cordeiro el al. [40] discussed the evolution of SDN from OpenFlow to data plane programmability. The survey briefly explained the layout of a P4 program and how it is mapped to the abstract forwarding model. It then listed various compilers, tools, simulators, and frameworks for P4 development. The authors categorized the literature into two categories: 1) programmable security and dependability management; 2) enhanced accounting and performance management. In the first category, the authors listed works pertaining to policy modeling, analysis, and verification, as well as intrusion detection and prevention, and network survivability. In the second category, the authors focused on network monitoring, traffic engineering, and load balancing. The surveys only lists a limited set of papers without explaining their main idea and how they are different from other works in the same category. Moreover, the survey was published in 2017, and based on Fig. 2, a significant percentage of P4-related works are missing.

Satapathy et al. [41] presented a short description about the pitfalls of traditional networks and the evolution of SDN. The report briefly described elements of the P4 language and demonstrates a sample program that achieves IP forwarding. The authors then discussed the control plane and P4Runtime [52] and enumerated three use cases of P4 applications that are provided by Barefoot Networks. The report concludes with potential future work expressed as research questions.

The short survey presented by Bifulco et al. [42] reviews the trends and issues of abstractions and architectures that realize programmable networks. The authors discussed the motivation of packet processing devices in the networking field and described the anatomy of a programmable switch. The proposed taxonomy categorizes the literature as state-based, abstraction-based, implementation-based, and layer-based. The layer-based consists of control/intent layer and data plane layer; the implementation-based encompasses software and hardware switches; the abstraction-based includes data flow graph and match-action pipelines; and the state-based differentiates between stateful and stateless data planes. The authors then proceeded with discussing open problems and issues that are related to finding the right abstraction for the data plane functionality.

Kaljic et al. [43] presented a survey on data plane flexibility and programmability in SDN networks. Similar to the aforementioned surveys, this work starts with the evolution and the migration from traditional to software-defined network architectures. The authors evaluated data plane architectures through several definitions of flexibility and programmability. In general, flexibility in SDN refers to the ability of the network to adapt its resources (e.g., changes in the topology or the network requirements). Afterwards, the authors identified key factors that influence the deviation from original data plane given with OpenFlow. Finally, the survey concludes with potential future research directions that address problems of programmability and flexibility in data planes.

Kannan et al. [44] presented a short survey related to the evolution of programmable networks. This work described

TABLE I
COMPARISON WITH RELATED SURVEYS.

Paper	P4 Language			Taxonomy			Workflow of P4 program			Discussions	
	Evolution	Description	Tutorial	Background	Literature	Comparison with legacy	Compilers and targets	Control plane	NOS	Challenges	Future directions
[38]	⦿	⦿	○	○	○	○	●	⦿	○	○	○
[39]	●	⦿	○	⦿	⦿	○	○	○	○	○	⦿
[40]	●	⦿	○	⦿	●	○	●	○	○	⦿	⦿
[41]	⦿	⦿	●	○	○	○	○	⦿	○	○	○
[42]	●	○	○	⦿	⦿	○	⦿	○	○	⦿	⦿
[43]	●	○	○	○	⦿	○	○	○	○	○	○
[44]	●	○	○	●	⦿	○	○	○	○	○	○
This paper	●	●	●	●	●	●	●	●	●	●	●

● Covered in this survey ○ Not covered in this survey ⦿ Partially covered in this survey

the pre-SDN model and the evolution to SDN and then to programmable data plane. The authors highlighted some features that programmable switches expose that were not available in fixed function devices. These features include stateful processing, accurate timing information, and flexible packet cloning and recirculation. The survey briefly described the match-action pipeline architecture and lists programmable ASIC vendors. The survey further categorized data plane applications into two categories, namely, network monitoring and in-network computing. While this work mentioned a considerable number of works belonging to these categories, it barely explained the operation and the main ideas of each work. The paper concludes with discussions and potential future work encompassing the aforementioned two categories.

Table I summarizes the topics and the features described in the related surveys. It also highlights how this paper differs from the existing surveys. To the best of the authors' knowledge, this work is the first to comprehensively and exhaustively explore the whole programmable data plane ecosystem. Specifically, this paper describes the P4 language (history and evolution from traditional networking and SDN, components, walkthrough program), while putting-forward a detailed taxonomy on the literature related to applications and how they are different from legacy approaches, the workflow of a P4 program (compiling, programming the control plane, network operating system, program testing), and challenges and future perspectives with respect to each category in the proposed taxonomy.

III. TRADITIONAL CONTROL PLANE AND SDN

A. Traditional Control Plane

Consider Fig. 3(a). With the traditional control plane approach, networks are connected using routing protocols such as Open Shortest Path First (OSPF) [53], ISIS [54], and the Border Gateway Protocol (BGP) [55]), running in the control plane *at each device*. Devices exchange routing information to build a local topology that converges to the global view of the network. Then, a routing algorithm (e.g., Dijkstra) acts upon the network state, computes the values used for forwarding decisions (e.g., output interface), and populates the local table. An example of such value is the output interface, used to forward a packet given a destination address. Note that both control and data planes are contained within a device.

With the traditional approach, both control and data planes are under full control of vendors and network functions are

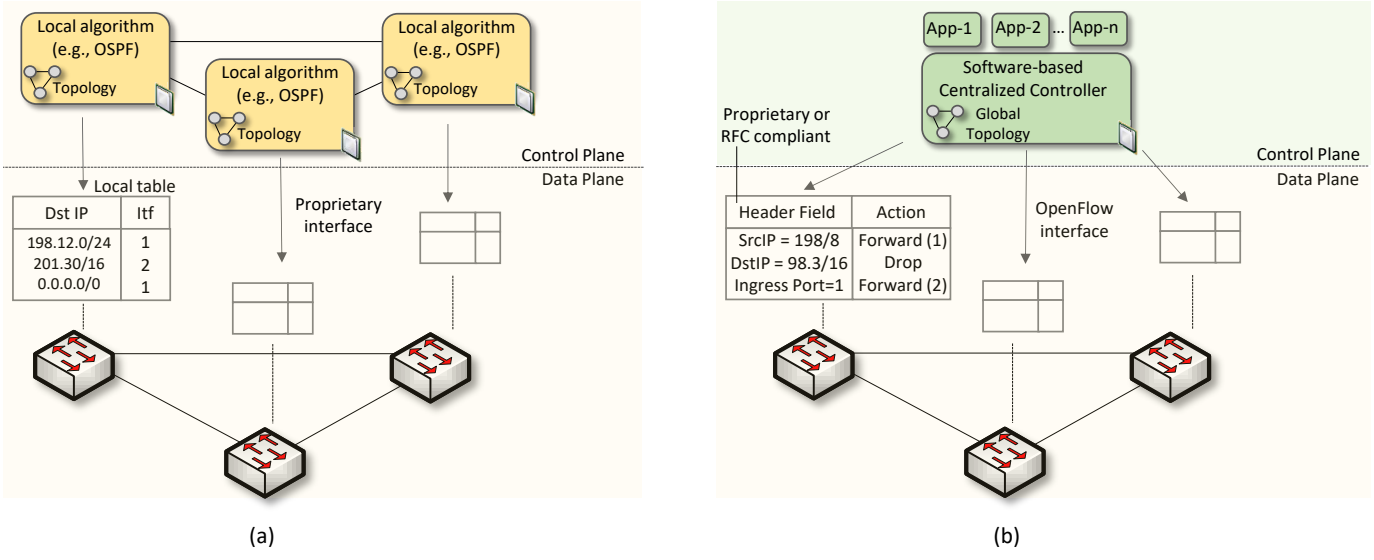


Fig. 3. (a) Traditional network. (b) SDN-based network.

fixed (either in the control software or hard-coded in the data plane). Also, although protocols are standardized, the control software is closed and proprietary, impeding the emergence of transformative innovation [56]. These fixed-function features are highlighted in yellow in Fig. 3(a).

One of the initial motivations for SDN was to reduce network complexity and management. Manufacturers were constantly adding features to satisfy a diverse set of customers. Thus, switches had to:

1) Incorporate diverse layer-2 features (e.g., Spanning Tree Protocol (STP), Virtual Trunking Protocol (VTP), Virtual Local Area Networks (VLANs), Ethernet, etc. for customers requiring layer-2 devices).
2) Include diverse layer-3 features (e.g., OSPF, Enhanced Interior Routing Protocol (EIGRP), BGP, etc. for customers requiring routing capability).
3) Incorporate multiple middle-box functionalities (e.g., NAT, Dynamic Host Configuration Protocol (DHCP), packet filtering) for users who need them.

To support the above functionalities, traditional switches require millions of lines of code, large power consumption, and tens to hundreds of people to manage a medium/large infrastructure. At the control plane, in order to locally run multiple algorithms simultaneously (e.g., running Dijkstra's algorithm to compute the STP topology at layer 2, running Dijkstra's algorithm to compute OSPF topology, and others) the devices use one or more control processors, as shown in Fig. 3(a).

Paradoxically, although the combined set of features required by different customers results in complex devices, most individual customers only use a handful of the features. In this context, in 2007, the Ethane project [7] proposed an alternative way of building and managing networks, giving birth to what is known as SDN. Ethane decoupled the control and data planes, enabling users to innovate by letting them develop software controllers that can remotely control thousands of devices and apply their own algorithms on the global topology view.

It is important to note that the traditional control plane approach is still widely spread and used by operators. Some reasons include the familiarity of network operators / IT engineers with this approach or the reluctance of organizations, for technical and/or cost reasons, to install complete new infrastructure. However, the transition from the traditional control plane approach to SDN control seems to continue unimpeded with the incremental replacing of traditional network devices by programmable devices [57].

B. SDN

According to McKeown [1], SDN can be considered as the first phase of the networking revolution that allows network operators (e.g., Internet Service Providers (ISP), campus networks, data center owners, and network owners in general) to speed up innovation and reduce cost and complexity of running their networks.

SDN delineates a clear separation between the control plane and the data plane, and consolidates the control plane so that a (single) centralized controller is capable of controlling multiple remote data planes, see Fig. 3(b). The controller, separated from the switches, computes the tables used by each switch and distributes them via a well-defined Application Programming Interface (API). The most popular protocol is Openflow [36]. The Ethane experiment [7] showed that a single Central Processing Unit (CPU) was capable of controlling and replacing 2,000 control CPUs in the switches, without negatively impacting the network.

SDN also generalized the traditional forwarding approach (based on destination addresses; e.g., destination IP address, destination MAC address) by observing that the forwarding process can be summarized as a two-step process: the *match* step where a destination address is matched against the local table, and the *action* step where the packet is sent to an egress port. With SDN, the match step can use other fields (other than destination IP/MAC addresses), protocols, switch attribute (e.g., ingress port) or combinations. For example, matching can be by the destination IP address and the destination TCP port, allowing separate forwarding for different TCP-based applications. Thus, SDN provides flexibility to implement policy-based routing, where forwarding is accomplished according to attributes specified by local administrative policy. The action step is generalized beyond simple forwarding and includes actions such as:

- Forwarding the packet out a specified single interface;
- Flooding the packet out a set of interfaces;
- Forwarding the packet to the controller;
- Modifying some field of the packet;
- Processing the packet at another (higher-numbered) flow table;
- Dropping the packet.

The match-action generalization enables packet switches to behave not only as a layer-3 router or layer-2 switch but also as firewalls, network address translators (NATs), load balancers, and other middle-boxes.

An essential aspect of controllers is its software implementation. By moving the controllers' implementation from the hands of vendors to the hands of developers and network owners, SDN fostered customization. Also, the ease of development, testing, and deployment of software made the innovation process more agile. Additionally, as these software implementations are open and publicly available (e.g., Open Network Operating System (ONOS) [58] and OpenDayLight [59] controllers), more programmers and professionals can contribute and add new features.

C. Motivation for Programmability at the Data Plane

Table II contrasts the main characteristics of traditional networks, SDN networks, and programmable-switch networks. SDN has been successful on shifting control of the control plane from vendors to operators by defining a clear separation of the control and data planes, and by providing an interface (e.g., OpenFlow) that permits the control plane to program and populate tables used by the data plane. SDN also provides an interface between the controller and third-party applications, whereas these applications can apply any algorithm against

TABLE II
FEATURES, TRADITIONAL NETWORKS, SDN NETWORKS, AND PROGRAMMABLE-SWITCH NETWORKS.

Feature	Traditional networks	SDN networks	Programmable-switch networks
Control - data plane separation	No clear separation	Well-defined separation	Well-defined separation
Control and data plane interface	Proprietary	Standardized APIs (e.g. OpenFlow)	Standardized (e.g., OpenFlow, P4Runtime) and program-dependent APIs
Control and data plane program-dependent APIs	NA/Proprietary	NA/Proprietary	Target independent
Functionality separation at control plane	No modular separation of functions	Modular separation: (1) functions to build topology view (state) and (2) algorithms to operate on network state	Same as SDN networks
Customization of control plane	No	Yes	Yes
Visibility of events at data plane	Low	Low	High
Flexibility to define and parse new fields and protocols	No flexible, fixed	Subject to OpenFlow extensions	Easy, programmable by user
Customization of data plane	No	No	Yes
ASIC packet processing complexity	High, hard-coded	High, hard-coded	Low, defined by user's source code
Data plane match-action stages	Proprietary	OpenFlow assumes in series match-action stages	In series and/or in parallel
Data plane actions	Protocol-dependent primitives	Protocol-dependent primitives	Protocol-independent primitives
Infield runtime reprogrammability	No	No	Yes

the global topology view exposed by the controller, wresting control of the network to the developers of the applications (network owner, third party, open source, researchers).

While SDN allows for the customization of the data plane, it is limited to the specifications of OpenFlow. With the latest specification (OpenFlow 1.5.1 [60]), the control plane can make decisions based on 45 header fields. Although the creative use of these fields permitted users to develop diverse applications, network owners still do not have full control of the packet processing in the data plane. For example, with network function virtualization and data center operations, there is an increasing need for new protocols to encapsulate packets (e.g., VXLAN [6], NVGRE [61]). Moreover, the process of standardizing and incorporating new protocols into the OpenFlow specification requires years. The need for a more flexible type of device that allows users to describe data plane operations and provide more visibility on packet behavior leads to what McKeown refers to as phase two of the networking revolution [1]. In this phase, network operators can control the software that defines how packets are processed in the data plane. Other advantages of programmable data planes include the program-dependent APIs, where the same P4 program running on different targets does not require modifications in the runtime applications (i.e., the control plane and the interface between control and data plane are target agnostic); the protocol-independent primitives used to process packets; the more powerful computation model where the match-action stages can not only be in series but also in parallel; and the infield reprogrammability of chips at runtime.

D. Analogy with other Domain Specific Processors

Historically, the data plane functionality such as packet parsing and its manipulation at line rate has been hard-coded by ASIC manufacturers through lengthy multi-year long processes. Similarly, the standardization and testing processes of new protocols have been dominated by vendors. As an example, by 2010, there were over 7,000 RFCs standards, many of which represented a barrier to entry for new players placed by vendors. Thus, a natural evolutionary question has been whether customized data plane functionality can also be implemented.

The conventional believe has been that functions at the data plane, operating at several terabits per second rates, must be hardcoded / backed into the silicon for performance. Under this model, only chip manufacturers can execute and implement new forwarding features and protocols; a process that takes years.

The above rigid model has been challenged in other computing domains [62]. The introduction of the general-purpose computers enabled programmers to develop applications running on CPUs. The use of high-level languages accelerated innovation by hiding the target hardware (e.g., x86, AMD). In signal processing, Digital Signal Processors (DSPs) were developed with instruction sets optimized for digital signal processing. Matlab is now widely used for developing new DSP applications. In graphics, Graphics Processing Units (GPUs) were developed in late 1990s and early 2000s with an efficient architecture and instruction sets to manipulate graphics and images. Open Computing Language (OpenCL) is one of the main languages to develop graphic applications. More recently, in machine learning, Tensor Processor Units (TPUs) and TensorFlow were developed in mid 2010s with instruction sets optimized for machine learning.

Over the last few years, a group of networking researchers conceived the idea of developing a machine model for networking, namely the Protocol Independently Switch Architecture (PISA) [62]. PISA is the machine original architecture optimized with instructions sets optimized for network operations, such as parsing packets, manipulating header fields, and executing operations on packets. PISA is to networking what DSP, GPU, and TPUs are to signal processing, graphics, and machine learning. Table III summarizes the analogy among

TABLE III
ANALOGY BETWEEN NETWORKS AND OTHER COMPUTING DOMAINS [62].

Domain	Processing Unit, Year	Main language/s
General computing	CPU, 1971[1]	C, Java, Phyton, etc.
Signal Processing	DSP, 1979	Matlab
Graphics	GPU, 1994	Open Computing Language
Machine Learning	TPU, 2015	TensorFlow
Computer Networks	PISA, 2016	P4

[1]First microprocessor introduced by Intel.

the different computing domains.

Since the conception of PISA, chip designers have built programmable PISA ASICs that operate at terabit speeds, making the data plane programmable. Performance reports indicate that programmable switches do not introduce performance penalty. On the contrary, these switches may produce higher throughput and lower latency and power consumption than fixed-function devices [63].

Programmable switches are now maturing as a technology that offers flexibility to network owners. Thus, while in an early phase where the workforce expertise is still low, programmable switches are used in unforeseen ways while increasing the pace of innovation, fostered by the agility of software development.

IV. PROGRAMMABLE SWITCHES

A. PISA Architecture

PISA is the original switch architecture that has enabled users to program the data plane, see Fig. 4. PISA is a packet processing model that has the following elements:

A.1. Programmable Parser

The programmable parser permits the programmer to define the headers (according to custom or standard protocols) and to describe how the switch should process those headers. The parser de-encapsulates the headers, converting the original packet into a parsed representation of the packet. The programmer declares the headers that should be recognized and their order in the packet. The parser can be represented as a state machine without cycles (direct acyclic graph), with one initial state (start) and two final states (accept or reject).

A.2. Programmable Match-action Pipeline

The programmable match-action pipeline executes the operations over the packet headers and intermediate results. A single match-action stage has multiple memory blocks (tables, registers) and Arithmetic Logic Units (ALUs), which allows for simultaneous lookups and actions. As some action results may be needed for further processing, stages are arranged sequentially.

A.3. Programmable Deparser

The deparser assembles the packet headers back and serializes it for transmission. Note that the PISA architecture is protocol independent; the programmer defines the headers and corresponding parser as well as actions executed in the match-action pipeline and deparser. The compiler then maps the program into the device.

A.4. Metadata Bus

The metadata bus moves intermediate results from one stage to another and thus is available in the entire pipeline. The metadata can be user-defined or intrinsic. User-defined metadata is defined as a structure in the P4 source code and is used to store user-defined data structures associated with each packet (e.g., protocol headers as they are parsed). The intrinsic metadata is used to interface with and control fixed-function components. It is provided by the device vendor and is used as predefined input for P4 components (e.g., input parameters to the parser, match-action units, or deparser). Examples of intrinsic metadata include: ingress port, egress port, timestamps, packet priority, and others [64].

In Fig. 4, the data plane program defines the format of the keys used for lookup operations. Keys can be formed using packet headers' information such as IP addresses, MAC addresses, ports, combinations of them, and other header fields. On the other hand, the control plane populates table entries with keys used to match packet information (e.g., destination IP address). The control plane also populates the table entries with action data or parameters that are used by the data plane if matches occur. Output results from previous match-action stages as well as tables are available to subsequent match-action stages [65].

Note the contrast between programmable switches and fixed-function switches. Fixed-function switches do not have the flexibility of reconfiguring the parser, operations, or deparser; their ASIC functionality is defined when the chip is designed. Thus, fixed-function switches only recognize a fixed set of protocols (e.g., Ethernet, IP, Multiprotocol Label Switching (MPLS)).

B. Programmable Switch Features

Programmable switches have several unique features that differentiate them from legacy fixed-function switches. Some of them are described by Shapiro [66] and include agility, top-down design, visibility, reduced complexity, and exclusivity and differentiation.

B.1. Agility

The initial goal of developing programmable switches has been to provide agility to the process of designing, testing, and adopting new protocols. Traditionally, this process has been dominated by chip manufacturers and taken several years from the initial design to the protocol implementation in switch ASICs. For example, the standardization process of VXLAN, which was officially specified in RFC 7348 in 2014 [67], took four years until the feature was finally available in ASICs [1]. With a PISA device, a programmer can design and test new protocols and features in minutes. The agile process is enabled by a top-down design.

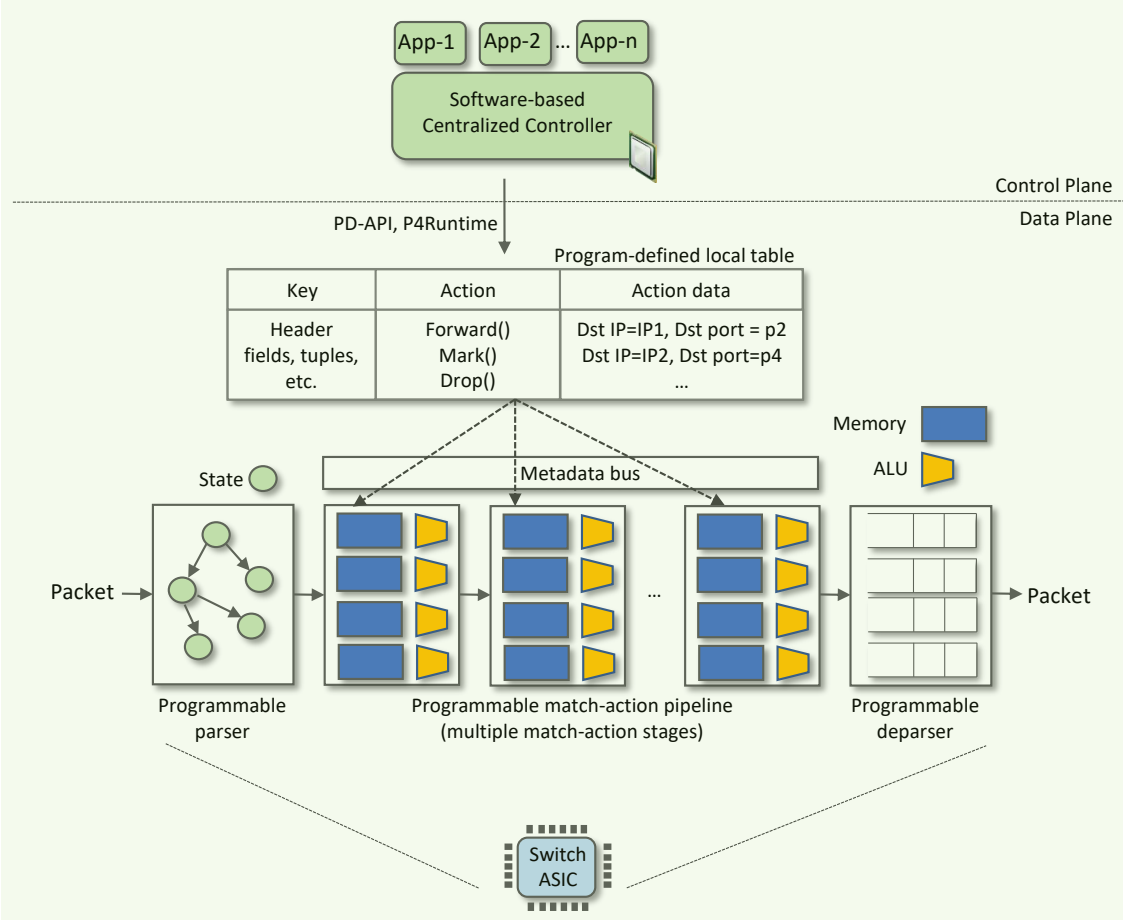


Fig. 4. Overview of the communication between the control plane and a PISA-based data plane.

B.2. Top-down Design

For several decades, the networking industry operated in a bottom-up approach. At the bottom of the system are the fixed-function ASICs, which enforce protocols, features, and processes available in the switch. Programmers and operators are limited to these capabilities when building their systems. As a consequence, systems have features defined by ASIC vendors that are rigid and may not fit the network operators' needs. Programmable switches and P4 represent a disruption of the networking industry by enabling a top-down approach for the design of network applications. With this approach, the programmer or network operator can precisely describe features and how packets are processed in the ASIC, using a high-level language, P4. The program is then compiled and downloaded into the ASIC, enabling a continuous evolution of protocols on the same hardware. The P4 program defines not only the data plane behavior but also how the control interacts with the data plane: the P4 program compilation also generates APIs which are used by the control plane to interact with the data plane (e.g., populating tables, reading registers).

B.3. Visibility

Programmable switches provide greater visibility into the behavior of the network. In-band Network Telemetry (INT) is an example of a framework to collect and retrieve precise information from the data plane, without intervention of the control plane [68]. INT defines a framework to collect precise information about individual packets and/or flows, such as switches and queues that the packet has traversed from source to destination, policies applied to the packets, and others. INT can be encapsulated into UDP, TCP, VXLAN, or even into other protocols or fields defined by the programmer, e.g., IP options header field. INT packets are generated by events defined by the programmer, which can be sent for analysis to a monitoring application. These include flow start or termination time, change of per-flow state (e.g. latency increase), queue depth and/or latency threshold breach, packet drop, or periodically as specified by the network operator.

Note that this flexibility enables the detection of events at a much more granular time scale (e.g., micro-bursts) than traditional products such as Netflow or sFlow. Initial INT applications include traffic monitoring, congestion analysis, and path and latency tracking. The greater visibility helps diagnose and fix failures faster.

B.4. Reduced Complexity

Fixed-function switches incorporate a large superset of protocols. These protocols consume resources and add complexity to the processing logic, which is hard-coded in silicon. Moreover, a network operator may only be interested in a small

TABLE IV
COMPARISON BETWEEN A P4 PROGRAMMABLE SWITCH AND A FIXED FUNCTION SWITCH [69].

Characteristic	P4 Switch (Tofino)	Fixed Function
Throughput	6.4Tb/s	6.4Tb/s
Number of 100G ports	64	64
Max forwarding rate	4.8B pps	4.2B pps
Max 25G/10G ports	256/258	128/130
Programmable	Yes (P4)	No
Power draw	4.2W per port	4.9W per port
Large scale NAT	Yes (100k)	No
Large scale stateful ACL	Yes (100k)	No
Large scale tunnels	Yes (192k)	No
Packet buffers	Unified	Segmented
LAG/ECMP	Full entropy, programmable	Hash seed, reduced entropy
ECMP	256-way	128-way
Telemetry	Line-rate per flow stats	SFlow (sampled)
Latency	Under 400 ns	Under 450ns

subset of protocols.

An advantage of programmable switches is the flexibility to implement only those protocols that are needed. Removing unused protocols frees resources that can be better utilized, reduces power consumption, minimizes the risk of software vulnerabilities and failures, and increases the reliability of the device. Key additional available resources that can be customized are Static RAM (SRAM) and Ternary Content-Addressable Memory (TCAM). SRAM is used for exact match tables, action data (see Fig. 4), counters and registers, while TCAMs are used for ternary and range match tables.

B.5. Exclusivity and Differentiation

With fixed-function devices, requesting changes or new features into the switch ASIC may demand months or years (if even viable) and will be costly. With programmable switches, a programmer can reprogram the ASIC and test the new design in hours. Moreover, the customized protocol or feature needs not to be shared with the chip manufacturer, which also spurs innovation.

Table IV shows a comparison between a P4-programmable switch (Tofino) and a fixed function switch [69]. The reported numbers are provided by [63], and were collected by partner Original Equipment Manufacturers (OEMs) of Barefoot Networks, an Intel Company. Both switches offer a throughput of 6.4Tb/s, and have 64 100G ports. The P4 switch offers 4.8 billion packets per second (pps), whereas the fixed-function switch offers 4.2 billion pps. Power consumption is better with Tofino with 4.2W per port, as opposed to 4.9W per port. The programmable switch offers new features due its capability of being programmable. Such features include NAT, stateful Access Control List (ACL), tunnels, unified packet buffers, full entropy, telemetry, etc. These features are not available in today's fixed-function devices. Finally, the minimum latency in Tofino is under 400ns, lower than the minimum latency in fixed function devices, 450ns.

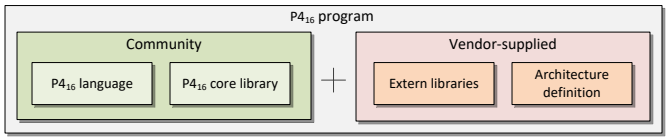


Fig. 5. $P4_{16}$ components separation [65].

V. P4 LANGUAGE

PISA is a domain-specific class of machine suitable for networking. P4, which stands for Programming Protocol-independent Packet Processors, is a language used to program PISA devices. P4 has a reduced set of instructions to express the actions used by the forwarding plane. These actions define the operations executed on packets; e.g., adding, removing, and modifying headers; manipulating and modifying fields; adding, subtracting, and shifting bits; and others. P4 has the following three goals:

- Reconfigurability: the parser and the processing logic can be redefined in the field to change the fundamental packet processing behavior.
- Protocol independence: the switch is protocol-agnostic (protocols are not hard-coded in the device). Instead, the programmer defines the protocols, the parser to extract headers, and the operations to process the headers.
- Portability and target independence: the details of the underlying switch are hidden from the P4 programmer. Instead, a compiler should take the switch's capabilities into account when turning a target-independent P4 program into a target-dependent binary.

A. $P4_{16}$ Motivation

The original specification of the P4 language was released in 2014, and is referred to as $P4_{14}$. In 2016, a new version of the language was drafted, which is referred to as $P4_{16}$. Both specifications use similar building blocks and concepts. $P4_{16}$ is a more mature language which extended the P4 language to broader underlying targets (ASICs, FPGAs, Network Interface Cards (NICs), software switches, etc.). The $P4_{16}$ language has significant syntax and semantics changes that are backwards-incompatible. $P4_{16}$ is also a simpler language than $P4_{14}$, as it only has 40 keywords while $P4_{14}$ has 70 keywords. Features such as counters, checksum units, and meters have been eliminated from $P4_{16}$ and moved into libraries.

The idea behind the development of $P4_{16}$ language is to separate the notions of a target from an architecture as shown in Fig. 5. A target is an embodiment of a specific hardware implementation, while an architecture is a set of P4-programmable components, externs, and fixed function components that are available in the device to the P4 programmer [65]. A main advantage of this separation is making the core language fairly small. The community (P4.org) provides the language which is common to any P4 program and contains all the core constructs, as well as the core library. Subsequently, every vendor providing their own platform (target implementing an architecture) supplies as part of their compiler back-end a library of their *extern* components, and the definition of the

architecture, which is the set of P4 programmable components that form the pipeline, and the interfaces between them. For example, a Tofino target uses the P4 language developed by the P4.org community while using the architecture definition and *extern* libraries provided by Barefoot Networks (vendor) as part of their Software Development Environment (SDE).

B. Elements of $P4_{16}$ Language

The elements of the $P4_{16}$ language are: data types, parser, controls, expressions, architecture description, and *extern* libraries. These elements are summarized below.

B.1. Data Types

$P4_{16}$ is a statically-typed language; hence, the type of a variable is known at compile-time instead of at run-time. P4's data types can be divided into two main categories: base types and derived types. The common base types include:

- *bit<n>*: unsigned integer (bitstring) of fixed width *n*. A single bit ($n = 1$) can be shortened as *bit*.
- *varbit<n>*: variable-length bitstring with a fixed maximum width *n*.
- *int<n>*: signed integer of size *n*, which is equal or greater than 2.
- *bool*: boolean values.

The base data types are used to construct derived types such as *headers*, *struct*, *header_union*, etc. Derived types are used in type declarations, where they introduce a new name for the type. The common derived types include:

- *header*: an ordered collection of members which are byte-aligned. P4 provides built-in operations to test and set the validity of a header: *isValid()*, *setValid()*, and *setInvalid()*. These are used in the match-action stages. The keyword *typedef* is used to define an alternative name for a type.
- *header_union*: combine multiple headers. For example, IPv4 and IPv6 headers can be combined in a *header_union* to represent the IP header.
- *struct*: similar to a C *struct*, is an unordered collection of members (with no alignment restrictions). It is used to introduces new types in the current scope.

Other useful types include *enum*, and header stacks which are used to create an array of headers.

B.2. Parser

The parser is a function that maps incoming packets (stream of bytes) into headers and metadata. It is written in a state-machine style with three predefined states: *start*, *accept*, and *reject*. Other states may be defined by the programmer. In each state, the program executes zero or more statements (mainly to extract headers and execute branching statements) and then transitions to another state. The parser always starts from the *start* state, and stops on either the *accept* state (the control is usually then transferred to the ingress block) or the *reject* state (which usually drops the packet). The *accept* and *reject* behaviors are defined by the provided architecture.

B.3. Controls

P4 Controls are used for both match-action processing and packet deparsers. In match-action processing, tables and their corresponding actions are defined.

- *tables*: in a match-action table, the program specifies the key (packet headers and/or metadata) to match on, the type of match (exact matching, Longest Prefix Match (LPM), or ternary), the possible actions to be executed when there is a match, the maximum size (how many entries can fit in the table), and a default action.
- *actions*: similar to loop-free void C functions, used to process a packet (e.g., modify headers). The data plane reads values written by the control plane through action parameters. The body of an action encompasses a sequence of statements and declarations.

Controls are essential for processing a packet. For example, a control for layer-3 forwarding may require a forwarding table that is indexed by the destination IP address, using LPM. The control may include actions to forward the packet when a hit occurs, and to drop a packet otherwise.

B.4. Expressions

Expressions are used to express basic operations and operators. Examples of expressions include assigning values to header fields within actions, a *select* expression used in the parser, etc.

B.5. Architecture Description

The architecture description is incorporated into the source code using the *include* directive header field, as a header file. The architecture description defines the programmable blocks that are available on the specific target device and the interfaces between blocks. The architecture description is provided in the form of a library P4 source file by the manufacturer of the device.

B.6. Extern Libraries

They provide support for specialized components such as counters, meters, register. Extern objects and functions are described using the extern construct, which describes the interfaces that such objects expose to the data plane. According to P4.org, the need for extern components is noted as platforms contain specialized components, which cannot be expressed in the core of the P4 language. By using externs, these components can be used through instantiation and method calling, even if the implementation is hidden [70].

C. P4 Basic Forwarding Program Walkthrough

This section demonstrates the building blocks for implementing a basic P4 IPv4 forwarding program. The program uses the V1Model architecture as it is open-source and implemented on top of BMv2, the behavioral open-source P4 software switch [46]. This program reflects the basic forwarding exercise found at [71].

C.1. V1Model

Fig. 6 depicts the V1Model architecture components. The V1Model architecture is implemented on top of BMv2's

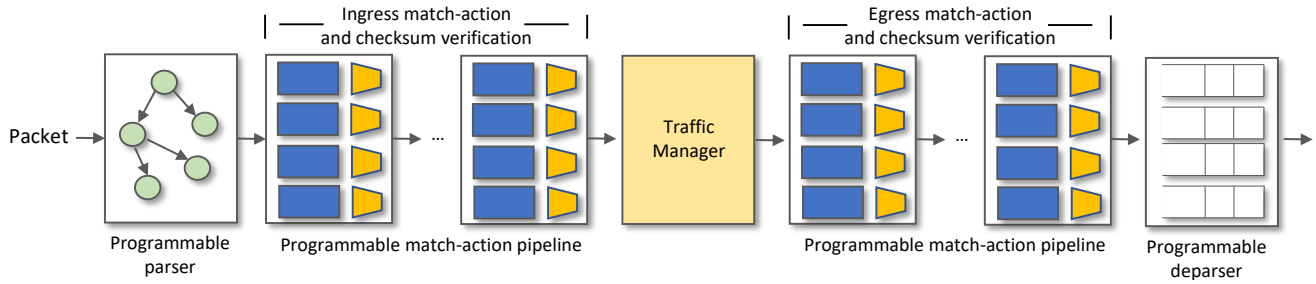


Fig. 6. V1Model architecture model.

simple_switch target, and consists of a programmable parser, ingress match-action processing block, traffic manager, checksum verification, egress match-action processing block, and a programmable deparser. The traffic manager schedules packets between input ports and output ports and performs packet replication.

C.2. V1's Standard Metadata

The standard metadata are used to control fixed-function components such as the traffic manager. Such metadata are specific to an architecture and provided by the device vendor. They are defined in a file that describes the target architecture (for example, v1model.p4 for V1Model architecture).

Fig. 7 lists the standard metadata of the V1Model. The most commonly used fields of the *standard_metadata* include: 1) *ingress_port*, which indicates the port on which the packet arrives, is set by the V1Model architecture before the packets arrives at the programmable parser; 2) *egress_spec*, which informs the traffic manager which port the packet should be sent to, is set within the ingress pipeline; and 3) *egress_port*, which indicates the port on which the packet is departing form, can be read only from the egress pipeline. There exist other metadata referred to as intrinsic metadata, that provide advanced features. For example, *enq_timestamp*, which is the timestamp (in microseconds) when the packet is first enqueued, *deq_timedelta*, which is the timestamp the packet spent in the queue, *deq_qdepth*, the depth (expressed in number of packets) of the queue when the packet was dequeued, and others. All the aforementioned metadata are read-only and can be accessed only from the egress pipeline.

```
1   struct standard_metadata_t {
2       bit<9> ingress_port;
3       bit<9> egress_spec;
4       bit<9> egress_port;
5       bit<32> instance_type;
6       bit<32> packet_length;
7       bit<32> enq_timestamp;
8       bit<19> enq_qdepth;
9       bit<32> deq_timedelta;
10      bit<19> deq_qdepth;
11      bit<48> ingress_global_timestamp;
12      bit<48> egress_global_timestamp;
13      bit<16> mcast_grp;
14      bit<16> egress_rid;
15      bit<1> checksum_error;
16      bit<3> priority;
17  }
```

Fig. 7. V1Model standard metadata.

```
1  #include <core.p4>
2  #include <v1model.p4>
3  const bit<16> TYPE_IPV4 = 0x800;
4
5  /************** H E A D E R S **************/
6
7  typedef bit<9>  egressSpec_t;
8  typedef bit<48> macAddr_t;
9  typedef bit<32> ip4Addr_t;

10 header ethernet_t {
11     macAddr_t dstAddr;
12     macAddr_t srcAddr;
13     bit<16>   etherType;
14 }

16 struct metadata {
17     /* empty */
18 }

20 struct headers {
21     ethernet_t   ethernet;
22     ipv4_t       ipv4;
23 }

10 header ipv4_t {
11     bit<4>    version;
12     bit<4>    ihl;
13     bit<8>    diffserv;
14     bit<16>   totalLen;
15     bit<16>   identification;
16     bit<3>    flags;
17     bit<13>   fragOffset;
18     bit<8>    ttl;
19     bit<8>    protocol;
20     bit<16>   hdrChecksum;
21     ip4Addr_t srcAddr;
22     ip4Addr_t dstAddr;
23 }
```

Fig. 8. Program headers.

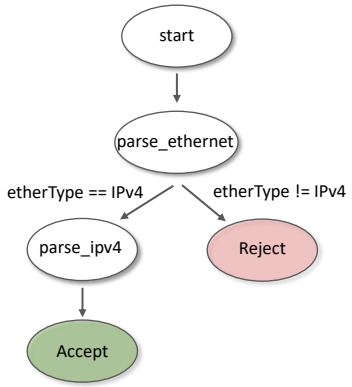
Note that this is specific to V1Model, and other architectures typically include different metadata with different accessing rules.

C.3. Program Headers and Definitions

Consider Fig. 8. The code starts by including the "core.p4" file (line 1) which defines some common types and variables used in all P4 programs. For instance, the *packet_in* and *packet_out* extern types which represent incoming and outgoing packets, respectively, are declared in core.p4. Next, the "v1model.p4" file is included (line 2) to define the V1Model architecture and all its externs used when writing P4 programs. Line 3 creates a 16-bit constant called *TYPE_IPV4* with the value 0x800. This means that *TYPE_IPV4* can be used later in the P4 program to reference the value 0x800. The typedef declarations used afterwards (lines 7 - 9) are used to assign alternative names to types. Subsequently, the headers and the metadata structs that are going to be used in the program are defined. These headers are customized depending on how the programmer wants the packets to be parsed. The program in Fig. 8 defines the Ethernet header (lines 10 – 14, column 1) [72] and the IPv4 header (lines 10 – 23, column 2). The declarations inside each header are usually written after referring to the standard specifications of the protocol. Note in the *ethernet_t* header the *macAddr_t* is used rather than

```
1   /********************** P A R S E R ************************/
2   parser MyParser(packet_in packet, out headers hdr,
3                           inout metadata meta,
4                           inout standard_metadata_t standard_metadata){
6       state start {
7               transition parse_ethernet;
8       }
9       state parse_ethernet {
10              packet.extract(hdr.ethernet);
11              transition select(hdr.ethernet.etherType) {
12                      TYPE_IPV4: parse_ipv4;
13                      default: reject;
14              }
15      }
16      state parse_ipv4 {
17              packet.extract(hdr.ipv4);
18              transition accept;
19      }
20  }
```

(a)



(b)

Fig. 9. Example of a custom parser. (a) P4 code and (b) corresponding state-machine diagram.

bit<48> due to the typedef declared in line 8. Lines 16 - 18 (column 1) show how to declare user-defined metadata, which are passed from one block to another as the packet propagates through the architecture. For simplicity, this program does not require any user metadata.

C.4. Parser

The parser can be considered as a state machine, where each state parses a header in the incoming packet, and transitions to another state. When the final state is reached, the control is transferred to the ingress pipeline block. Fig. 9 demonstrates how a P4 parser is mapped to a state machine diagram. The parser always starts with the initial state called *start*. Two headers were defined in the basic forwarding program: Ethernet and IPv4; therefore, two states are created: *parse_ethernet* and *parse_ipv4*. In the *start* state, the program unconditionally transitions (line 7) to the *parse_ethernet* state, and the Ethernet header is then extracted from the packet (line 10). Note that *packet* is an instance of the *packet_in* extern (specific to V1Model), and is passed as a parameter to the parser. The *extract* method associated with the packet extracts N bits, where N is the total number of bits defined in the corresponding header (for example, 112 bits for Ethernet). Afterwards, the etherType field of the Ethernet header is examined using the select statement, and the program branches to the *parse_ipv4* state if the etherType field corresponds to IPv4. Otherwise, the state transitions to the accept state, which moves the control to the ingress pipeline block. In the *parse_ipv4* state, the IPv4 header is extracted and the program unconditionally transitions to the accept state.

C.5. Ingress Control Block

In the ingress control block, tables and their actions are defined. Tables describe the match-action units, where a lookup is performed on a key ("match" phase), and an action is executed to modify the input data ("action" phase). Fig. 10 shows the ingress control block portion of the P4 program. Two actions are defined, *drop* and *ipv4_forward*. The *drop* action (lines 5 - 7) essentially invokes the *mark_to_drop* primitive, causing the packet to be dropped at the end of the ingress processing. The *ipv4_forward* action (lines 9 - 14)

```
1   /********** I N G R E S S P R O C E S S I N G **********/
2   control l3_forwarding(inout headers hdr,
3                         inout metadata meta,
4                         inout standard_metadata_t standard_metadata) {
5       action drop() {
6           mark_to_drop(standard_metadata);
7       }
9       action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
10          standard_metadata.egress_spec = port;
11          hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
12          hdr.ethernet.dstAddr = dstAddr;
13          hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
14      }
15      table ipv4_lpm {
16          key = {
17              hdr.ipv4.dstAddr: lpm;
18          }
19          actions = {
20              ipv4_forward;
21              drop;
22              NoAction;
23          }
24          size = 1024;
25          default_action = drop();
26      }
27      apply {
28          if (hdr.ipv4.isValid()) {
29              ipv4_lpm.apply();
30          }
31      }
32  }
```

Fig. 10. Ingress block code.

accepts as input the destination mac address and an egress port. These parameters are inserted by the control plane, and updated in the packet during the ingress processing. Line 10 assigns the new egress port to the *standard_metadata's* egress port field (the field that the traffic manager looks at to determine which port the packet should be sent from). Line 11 assigns the original destination MAC address found in the packet to the new source address. Line 12 assigns the destination MAC address passed as parameter to the packet's new destination address. Line 13 decrements the Time-to-Live (TTL) field of the IPv4 header as the program implements a basic IPv4 router.

Lines 15-26 implement a table named *ipv4_lpm*. The table is matching against the destination IP address inside the header using the longest prefix match. The actions associated with the table are *ipv4_forward*, *drop*, and *NoAction*. The default action which is invoked when there is a miss is *drop*. The

```
/************ C H E C K S U M C O M P U T A T I O N ************/
control MyComputeChecksum(inout headers hdr, inout metadata meta){
     apply {
       update_checksum(
           hdr.ipv4.isValid(),
            { hdr.ipv4.version,
              hdr.ipv4.ihl,
              hdr.ipv4.diffserv,
              hdr.ipv4.totalLen,
              hdr.ipv4.identification,
              hdr.ipv4.flags,
              hdr.ipv4.fragOffset,
              hdr.ipv4.ttl,
              hdr.ipv4.protocol,
              hdr.ipv4.srcAddr,
              hdr.ipv4.dstAddr },
           hdr.ipv4.hdrChecksum,
           HashAlgorithm.csum16);
     }
}
```

Fig. 11. Checksum computation's code.

maximum number of entries a table can support is configured manually by the programmer (line 24). Note however that the number of entries is limited by the amount of storage in the switch's SRAM.
The control block starts executing from the *apply* statement which contains the control logic. In this program, the *ipv4_lpm* table is activated in case the incoming packet has a valid IPv4 header.

C.6. Compute Checksum

In the ingress pipeline block, the TTL value in the IPv4 header was decremented as the program is simulating an IPv4 router. Therefore, the IPv4's checksum must be recomputed. In the V1Model architecture, the extern *update_checksum* is used to compute the checksum of a specific header. It accepts as input the headers' fields, the checksum field to be computed, and the checksum algorithm. The checksum is computed only if the corresponding header is valid. The checksum code is listed in Fig. 11.

C.7. Deparser

The deparser code is shown in Fig. 12. The packet is accepted in the parser's parameters as an instance of *packet_in*. The deparser on the other hand has a *packet_out* type in its parameters. The *packet_out* type includes the *emit* method which accepts the headers to be reassembled when the deparser constructs the outgoing packet. Note that the order of emitting packets' headers is important, and the headers are only emitted in case they are valid. For example, the IPv4 header will not get emitted if an incoming packet does not have an IPv4 header (its IPv4 header is invalid).

```
/****************** D E P A R S E R ******************/
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
    }
}
```

Fig. 12. Deparser's code.

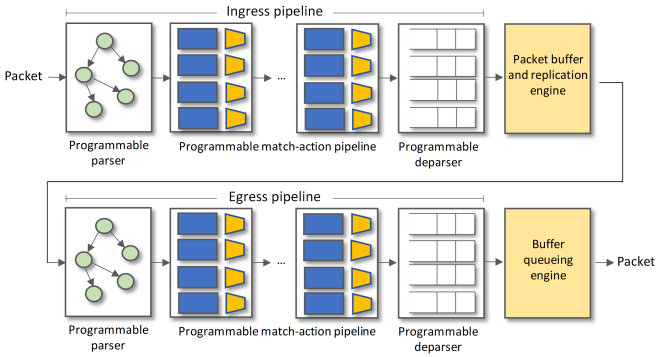


Fig. 13. Portable switch architecture model.

D. P4_16 and the Portable Switch Architecture

$P4_{14}$ was developed to program PISA-like devices [83]. Since then, multiple programmable devices and targets have been developed, which was one of the motivations to develop $P4_{16}$.

One of the main innovations of $P4_{16}$ is the introduction of the P4 architecture described above. The P4 Architecture Working Group [84] describes the P4 architecture as the equivalent of a programming model, since it allows a programmer to write P4 code against a well defined set of functionality defined by the vendor. The P4 Architecture Working Group has also provided the specification for the Portable Switch Architecture (PSA). The PSA is a reference switch architecture, intended for portability that decouples P4 programs from the underlying target. The PSA [85] is developed by the P4 Architecture Working Group and describes common switch capabilities. It extends capabilities beyond match-action tables (e.g., stateful registers, meters) and identifies six *P4-programable* blocks and two fixed-function *configurable* blocks.

The PSA model is shown in Fig. 13 [85]. The model has two identical pipelines: ingress and egress. The two configurable (non-programmable) blocks are shown in yellow: Packet Buffer and Replication Engine (PBRE) and Buffer Queueing Engine (BQE). Both configurable blocks implement queueing functionality. The PBRE also implements packet replication functionality, which is used for multicasting packets. Note that currently P4 does not support programmability of the PBRE and BQE, therefore Quality of Service (QoS) and other queueing-related aspects are beyond the P4 specifications.

E. P4 and Other Architectures

Since P4 and the PISA architecture were originally conceived, multiple P4 architectures were developed. Table V lists some relevant architectures. The V1model is the architecture used in the $P4_{14}$ language specification.

The Portable NIC Architecture (PNA) is a reference model for network interface cards [76]. The PNA blocks are similar to those of the PSA, but extended for NICs. The Flex Switch Abstract Interface (SAI) is a hybrid architecture [86], as it mixes programmable and fixed (configurable) blocks which conform the pipeline. In a hybrid pipeline, some fixed blocks

TABLE V
P4 ARCHITECTURES AND MODELS.

Architecture	Use	Developer
PISA	Reference P4 device; some vendors use the PISA model, e.g., a Barefoot's Tofino [26], Cisco Doppler, Intel Flexpipe [73]. $P4_{14}$ targeted PISA-like devices	P4.org
PSA	Target architecture with the goal of becoming a switch architecture that may be implemented on any switch target [74]	P4.org Architecture Working Group
BMv2	Flexible framework, it enables developers implement their own architecture as a software switch; used in software switches for rapid prototyping, learning [46]	P4.org
V1model	Reference, used in $P4_{14}$ language specification [46]. It can be implemented on top of BMv2 [75]	P4.org
Portable NIC Architecture (PNA)	Analogous to PSA (see Fig. 13), but for NICs [76]; the targets are mostly NICs; it complements PSA specification	P4.org Architecture Working Group
Flex Switch Abstraction Interface (SAI) [77]	Hybrid architecture; it mixes programmable and configurable blocks conforming the pipeline. Used in production switches / networks; e.g., white-box switches with SONiC operating system	Microsoft, Open Compute Project (OCP)
SimpleSumeSwitch	Architecture used by the FPGA device board called NetFPGA SUME [78–81]. The architecture is also supported by BMv2	Digilent, the University of Cambridge, Stanford University
Vendor specific	Proprietary architectures; e.g., Tofino Native Architecture (TNA) [82]	Vendors

may implement protocols such as Ethernet, IP, and ARP, which facilitates the use of these protocols in the pipeline, reducing the development time. Flex SAI is designed to work with SAI, which is a software module running on the operating system that provides a vendor-independent API to control the forwarding elements. SAI is incorporated into SONiC [87], a Linux operating system that runs on white-box switches (e.g., Edgecore). Initially developed by Microsoft, SAI is now an open-source project managed by the Open Compute Project (OCP) [88]. OCP is an organization composed of multiple companies interested in data center products, such as Facebook, Intel, Google, Microsoft, and others.

The SimpleSumeSwitch architecture is the P4 architecture defined for the NetFPGA SUME board, which is an FPGA-based Peripheral Component Interconnect (PCI) Express board. The architecture consists of a single parser, single match-action pipeline, and single deparser [80]. NetFPGA SUME is used for rapid prototyping of 10 Gbps and 40 Gbps applications [78].

Additionally, there are other vendor specific architectures, such as Barefoot's Tofino Native Architecture [82]. P4 programs are portable across different targets as long as the vendor provides the compiler for a specific architecture [89].

F. P4 and Fixed-function Devices

Functions such as forwarding based on Ethernet (layer 3) and IP addresses (layer 3) are easily described in P4. These functions are commonly implemented by fixed-function devices as well. Fixed-function device vendors are increasingly exploiting this flexibility of describing the pipeline and are using P4 to describe data plane elements; using P4 to map logical resources (e.g., tables) to hardware resources in a unified manner enables code portability.

G. NOS and Open Networking

Legacy switches are often equipped with proprietary Network Operating Systems (NOS) provided by the vendors. Proprietary NOSs are popular, thoroughly tested, and easy to deploy in today's network infrastructure. However, they are costly and non-scalable as changes in the devices require vendor's intervention. Recent trends in the networking industry are shifting towards open networking, a term that suggests deploying white box switches (e.g., Edgecore [90], Quanta [91], Mellanox [92]) running open networking software (e.g., Microsoft's SONiC [87], Open Network Linux (ONL) [93], Facebook Open Switching System (FBOSS) [94]). In the context of P4, the programmable chip vendor (e.g., Barefoot) provides the software development toolchain and its binaries. The programmer then compiles a custom P4 code, and deploys the binary output to the switch daemon running on top of the NOS (e.g., ONL).

Note that not all operators have the technical skills to write production-grade P4 programs for white box switches. ASIC vendors typically provide the source code for *switch.p4*, an implementation of various networking features needed for typical cloud data centers, including Layer 2/3 functionalities, ACL, QoS, etc. The operators can simply compile the switch.p4 program using the ASIC SDE and deploy the program to the NOS. As an example, Barefoot provides their implementation of switch.p4 and allows operators to integrate it with SONiC through SAI [86]. SAI is used to define the interfaces between the forwarding element (ASIC) and the control plane. NOSs such as SONiC use open-source stacks (e.g., Free Range Routing (FRRouting) [95]) to enable routing and to populate the forwarding tables in the ASIC.

VI. WORKFLOW OF A P4 DESIGN

Programming a P4 switch, whether a hardware or software implementation, requires a software development environment that includes a compiler. Cascone [85] provides details on the P4 workflow implementation covering the software development. Consider Fig. 14. The compiler maps the target-independent P4 source code (mycode.p4) to a specific platform. The compiler, architecture model, and the target device are vendor specific and provided by the vendor. The P4 source code and controller are customized by the user.

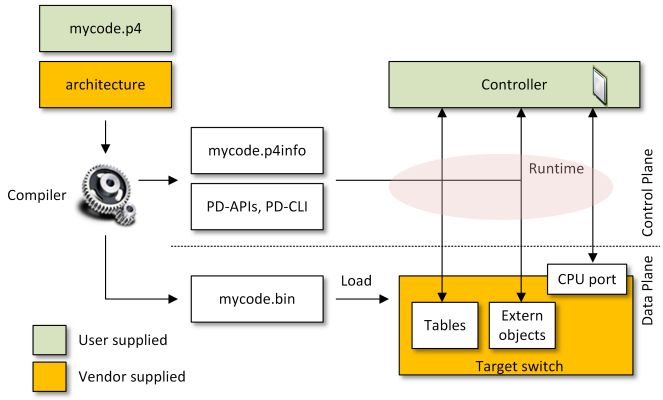


Fig. 14. Workflow design [85]. The compiler, architecture model, target switch ASIC, and the compiler are provided by the vendor of the device. The P4 source code and controller are customized by the user. The compiler generates the binary file to be loaded into the target, and PD APIs used by the control plane to communicate with the data plane at runtime. The data plane has a port used to communicate with the control plane's CPU.

The compiler generates a binary file (mycode.bin) that is loaded into the device, which includes the instructions and resource mappings. The compiler also generates Program-Dependent (PD) runtime APIs and Command-Line Interface (CLI) commands. The APIs and CLI are used by the control plane / user to interact with the data plane. Examples include adding/removing entries into match-action tables, retrieving data from the data plane (statistics, meters), and configuring/modifying actions when matches occur. The file p4info file (mycode.p4info) contains the information needed by the controller to manipulate tables and objects from the data plane, such as the identifier of the tables, fields used for matches, keys, action parameters, and others. This file is used by the controller to communicate with the data plane using P4Runtime. P4Runtime [52] is a control plane specification for controlling the data plane elements of a device defined or described by a P4 program.

The programmable device has a specific port for sending packets to the control plane's CPU. A packet directed to the CPU follows exactly the same processes across the data plane as other packets (parser, match-action pipeline, and deparser) before being forwarded to that port. The CPU port is the main channel used by the control plane to inform the data plane of events that require control plane actions (e.g., updating a table based on the received packet).

A. Source Code

Programming in P4 is similar to programming in other languages, such as C. The source code must at least possess the following two header files: core.p4 and architecture.p4 (or an equivalent file describing the architecture). The file core.p4 is the main library and contains built-in constructs. The architecture file must be provided by the vendor of the target device.

B. Compiler

The compiler maps the P4 source code to the target platform, allocating resources such as tables (TCAMs, SRAM), counters, registers, and others needed by the program. Some of the goals of the $P4_{16}$ reference compiler are to [103]:

- Provide an open source front-end. Here, front-end refers to the $P4_{16}$ language and constructs that are target-independent. The front-end does not include any target-specific information;
- Support current and future versions of P4 code;
- Support multiple back-end devices. Back-ends refer to the targets, such as hardware ASICs, NICs, FPGAs, and software switches;
- Integrate to other tools (e.g., software-development environments, debuggers);
- Provide support for extensibility. This facilitates the incorporation of specific device capabilities, using externs;

The implementation of the front-end, P4 reference compiler is called *p4c* [45]. The *p4c* front-end currently supports several back-ends [104]. Table VI lists some P4 compilers (some of which incorporate *p4c*) with their respective targets.

C. Compiler's Output Files

The open-source P4 compiler (p4c) generates two outputs: a target-specific binary and a *p4info* file. The target-specific output can be a binary configuration for ASIC (e.g., mycode.bin deployed on Tofino ASIC), bitstream for FPGAs, JavaScript Object Notation (JSON) file for software switches (e.g., mycode.json for BMv2), etc. The *p4info* file represents a schema of the pipeline for runtime control purposes. It contains the P4 program's attributes such as tables, match fields, actions, parameters, etc. It is based on Protobuf [105], which makes it easy to be deserialized and consumed by the control plane. The p4info file is target-independent, hence, the same file can be used by different targets such as software

TABLE VI
P4 TARGETS AND COMPILERS.

Compiler	Target	Architecture
p4c [45]	P4 reference compiler; it provides support for multiple targets (e.g., BMv2, simple-switch model)	Support for multiple architectures (e.g., V1model, PSA)
Xilinx SDNet [96]	FPGA	SimpleSumeSwitch
TC P4 Compiler [97]	Linux	Linux Target Architecture
P4-to-VHDL [98] [99]	FPGA	Vendor specific
P4-eBPF [100] [101]	Linux kernel	
p4c-sai [77]	Flex SAI devices	Flex SAI
Other proprietary compilers (e.g., Barefoot's p4c-tofino, Netronome's compiler) [102]	Vendor specific	Vendor specific (ASICs, NICs)

```
action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
  standard_metadata.egress_spec = port;
  hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
  hdr.ethernet.dstAddr = dstAddr;
  hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}
table ipv4_lpm {
  key = {
      hdr.ipv4.dstAddr: lpm;
  }
  actions = {
      ipv4_forward;
      drop;
      NoAction;
  }
  size = 1024;
  default_action = drop();
}
```

(a) mycode.p4

```
tables {
  preamble {
    id: 33574068
    name: "MyIngress.ipv4_lpm"
    alias: "ipv4_lpm"
  }
  match_fields {
    id: 1
    name: "hdr.ipv4.dstAddr"
    bitwidth: 32
    match_type: LPM
  }
  action_refs {
    id: 16799317
  }
  size: 1024
  ...
actions {
  preamble {
    id: 16799317
    name: "MyIngress.ipv4_forward"
    alias: "ipv4_forward"
  }
  params {
    id: 1
    name: "dstAddr"
    bitwidth: 48
  }
  params {
    id: 2
    name: "port"
    bitwidth: 9
  }
```

(b) mycode.p4info

Fig. 15. Example of a *p4info* file. (a) P4 code and (b) corresponding generated *p4info* file.

MyIngress.ipv4_lpm

Key	Match – Action	Action Parameters
hdr.ipv4.dstAddr	192.168.100.1/32 – MyIngress.ipv4_forward()	11:22:33:44:55:66, 1

⇩

```
table_id: 33574068 ("MyIngress.ipv4_lpm")
match {
  field_id: 1 ("hdr.ipv4.dstAddr")
  lpm {
    value: "\\xc0\\xa8\\x64\\x01"
    prefix_len: 32
  }
}
action {
  action {
    action_id: 16799317 ("MyIngress.ipv4_forward")
    params {
      param_id: 1 ("dstAddr")
      value: "\\x11\\x22\\x33\\x44\\x55\\x66"
    }
    params {
      param_id: 2 ("port")
      value: "\\x00\\x01"
    }
  }
}
```

Fig. 16. Protobuf representation of an entry insertion message.

switches, ASICs, etc. Fig. 15 demonstrates an excerpt of a P4 program and its corresponding generated p4info file. The P4 program contains an IPv4 table *ipv4_lpm* matching on the destination IP of the packet using LPM, and has an action *ipv4_forward* which accepts as parameters the destination MAC address and the egress port. This information is encoded in the p4info file (corresponding words are colored in green). Note however that actions' implementations are not written in the p4info file. Fig. 16 shows the Protobuf representation of an entry insertion message.

D. P4Runtime

Once a P4 program is compiled into a target-specific configuration binary, it is loaded into the data plane of the device. A control plane is then needed to populate the tables defined in the P4 program. P4Runtime is a control plane API for controlling the data plane of a device defined by a P4 program. It can be used to insert, update, and delete entries in P4 tables as well as controlling other entities of the program such as counters, meters, etc. The main motivation behind P4Runtime is that no existing approach for controlling the data plane is both protocol-independent and target-independent. Table VII compares the existing approaches for controlling the data plane.

1) The autogenerated P4 compiler (p4c) runtime APIs define methods to add/modify/delete entries for each table defined in the P4 program. This approach is protocol-independent as it is based on P4, but it is program-dependent and requires restarting the control plane to load a new program on the data plane.
2) BMv2 CLI provides a command-line interface to manipulate the P4 tables. It is program-independent, but target-specific. Therefore, the control plane written for BMv2 cannot be ported to other targets (e.g., Tofino).
3) OpenFlow: Initially designed to be a target-independent protocol for manipulating the data plane. However, OpenFlow is protocol-dependent as the headers and the actions are hardcoded in the specifications. In order to support a new protocol, the OpenFlow specifications must be extended. Moreover, adding support to a new protocol requires recompiling the software stack.
4) SAI is target-independent, but was designed with the goal of providing abstraction for the legacy forwarding pipelines (L2, L3, ACL); therefore, SAI is protocol-dependent.

P4Runtime is both protocol-independent and target-independent. It is a Protobuf-based API [105], hence it is efficient in wire format, and its code used to serialize and deserialize messages is automatically generated. P4Runtime uses gRPC transport, which automatically generates client and server stubs, and provides authenticated and encrypted bi-directional stream channels for exchanging Protobuf messages. P4Runtime allows pushing a new P4 program to the data plane to reconfigure the pipeline in field, without the need to recompile the control plane software stack. It can be used as a remote control plane and as a local control plane.

E. Packet Test Framework (PTF)

Packet Test Framework (PTF) [107] is used for automated testing for both the data plane programs as well as the control plane APIs. It is a universal framework that allows testing the behavior of the data plane program (e.g., verify that the packet is received on the right port). It also allows accessing

TABLE VII
PROPERTIES OF A RUNTIME CONTROL API [106].

API	Protocol independent	Target independent
Auto-generated (p4c)	×	✓
BMv2	✓	×
OpenFlow	×	✓
SAI	×	✓
P4Runtime	✓	✓

the data plane through the API to check the behavior of the data plane state (e.g., check if the correct counter/register is modified). Moreover, it allows testing the API behavior (e.g., check whether the table entries were inserted and the time it takes to insert an entry, etc.) Another important characteristic of PTF is that it allows P4 developers to write formal support requests (questions or bugs). This allows the support team to understand what the program does and what is to be expected from the program. As a result, it allows P4 programs to be easily reproducible, automated, and shared.

PTF is based on the standard Python's *unittest* module and uses python API's bindings. It also uses Scapy [108] to generate packets. PTF is maintained by the P4.org community and the source code is available at [107].

Some proprietary devices (e.g., Barefoot's Tofino) further simplify the debugging by allowing the developer to manage snapshots in the pipeline. This feature is extremely helpful as it allows tracing metadata values through any range of Match-Action Unit (MAU) stages.

F. Next-Generation SDN (NG-SDN)

Fig. 17 illustrates the stack components of the Next-Generation SDN (NG-SDN) [109]:

- ONOS/μONOS controller [58]: SDN controller that supports the NG-SDN interfaces (P4Runtime [52], gNMI [110], gNOI [111]) and adopts a OpenConfig-based management subsystem [112]. It is based on cloud native (e.g, Kubernetes [113]) and enables high-level applications to push information to the silicon.
- Forwarding devices: those can be devices that support full P4 programmability or fixed-function devices with partial programmability.
- Stratum: the switches run Stratum [114], a lightweight NOS that implements these NG-SDN interfaces and is designed to be both switch vendor and ASIC vendor agnostic. Stratum is a thin switch NOS since it does not necessary implement all the necessary SDN building blocks found in other solutions.

A set of services responsible for control-plane management and configuration, monitoring and telemetry, and administration and orchestration are part of the control and management plane in the ONOS controller.

There will be a contract between the controller and the P4-based switches for the various rules, depending on the profile (e.g., leaf switch, spine switch). Various interfaces exist between the controller and the switches. First, the interface that controls the programmable and the fixed elements of the pipeline is the Protobuf-based P4Runtime (see Section VI-D).

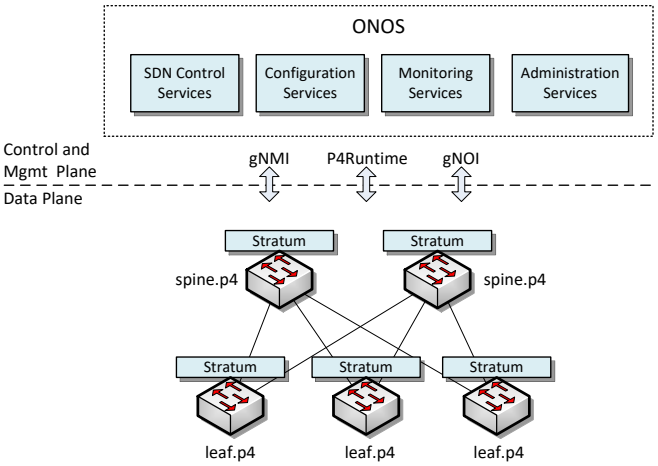


Fig. 17. Next-Generation SDN (NG-SDN) stack components in a leaf-spine topology.

Second, the interface for configuration and streaming telemetry is gRPC Network Management Interface (gNMI) which uses the OpenConfig model set [112]. gNMI is used to manipulate network device configuration (install, change, delete) and to view operational data [110] by using YANG configuration modeling language [115]. Third, the gRPC Runtime Operations Interface (gNOI) [111] which has various microservices for operations / runtime management (e.g., rebooting device, ephemeral state management, managing SSL keys, etc.).

VII. SURVEY METHODOLOGY

This section starts with the survey portion of this paper. Specifically, this section describes the employed systematic methodology that was adopted to generate the proposed taxonomy. The results of this literature survey represent derived findings by thoroughly exploring more than 130 data plane-related research works starting from 2016 up to August 2020. The distribution of which is summarized in Fig. 18.

The proposed taxonomy is demonstrated in Fig. 19. The taxonomy was meticulously designed to cover most significant works related to data plane programmability and P4. The aim is to categorize the surveyed works based on various high-level disciplines. The proposed taxonomy provides a clear separation of categories so that a reader interested in a specific discipline can only read the works pertaining to said discipline. The correctness of the taxonomy was verified by

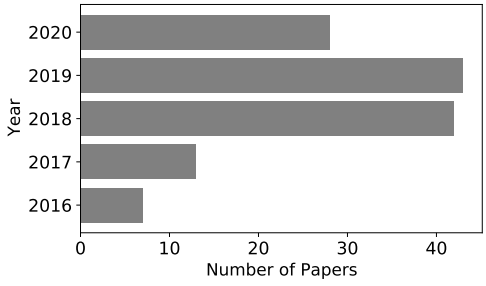


Fig. 18. Distribution of surveyed data plane research works per year.

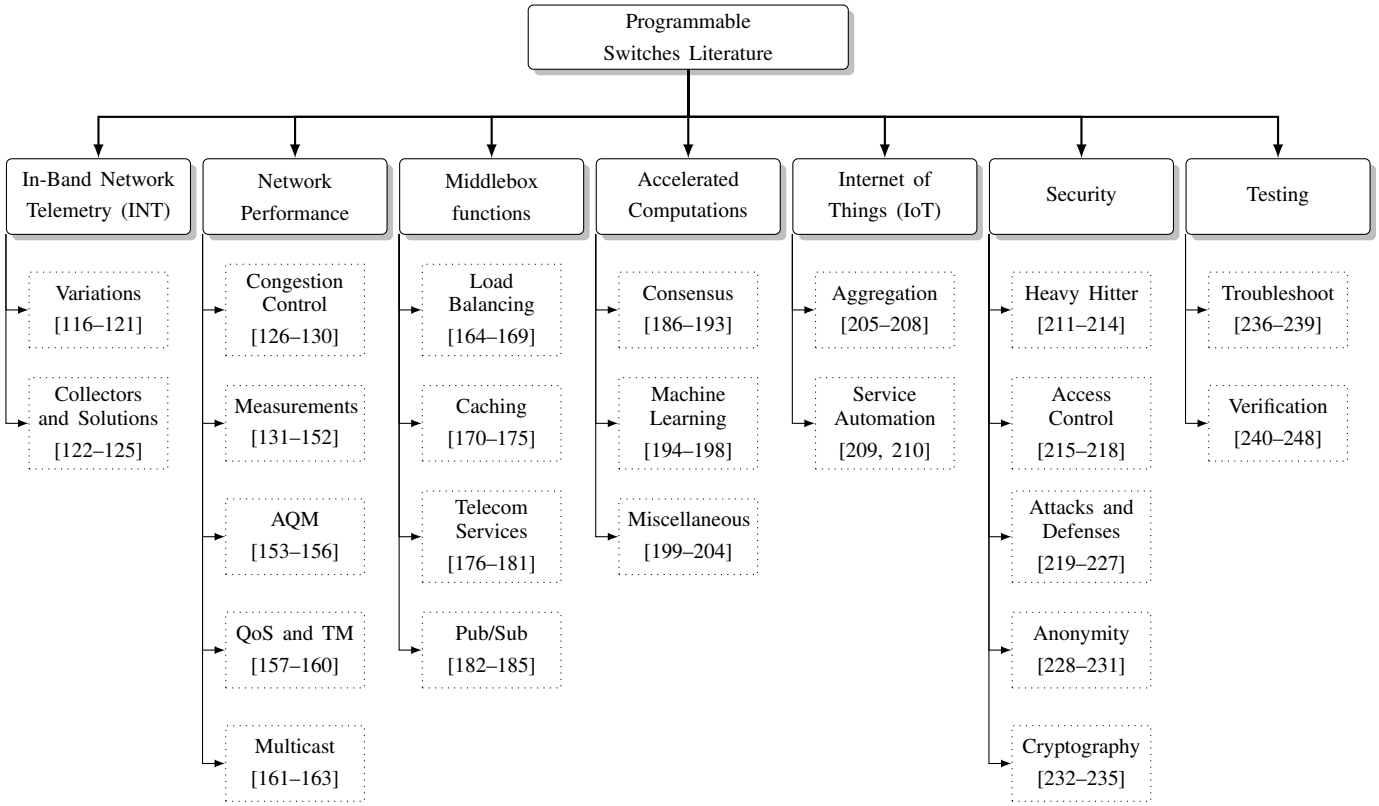


Fig. 19. Taxonomy of programmable switches literature based upon relevant, explored research areas.

carefully examining the related work of each paper to correlate them into high-level categories. Each high-level category is further divided into sub-categories. For instance, various measurements works belong to the sub-category "Measurements" under the high-level category "Network Performance".

The survey compares the results and the features offered by programmable data plane approaches with those of the contemporary and legacy ones. This detailed comparison is elaborated upon for each sub-category, giving the interested reader a comprehensive view of the state-of-the-art findings of that sub-category. Additionally, the survey presents various challenges and considerations, as well as current and future directions for each of the sub-categories in the taxonomy.

VIII. In-band Network Telemetry (INT)

Conventional monitoring and collecting tools and protocols (e.g., ping, traceroute, Simple Network Management Protocol (SNMP), NetFlow, sFlow) are by no means sufficiently accurate to troubleshoot the network, especially with the presence of congestion. These methods provide milliseconds accuracy at best and cannot capture events that happen on microseconds magnitude. Moreover, they cannot provide per-packet visibility across the network.

In-band Network Telemetry (INT) [249] is one of the earliest key applications of programmable data plane switches. It enables querying the internal state of the switch and provides fine-grained and precise telemetry measurements (e.g., queue occupancy, link utilization, queuing latency, etc.). INT handles events that occur on microseconds scale, also known as *microbursts*. Collecting and reporting the network state is performed entirely by the data plane, without any intervention from the control plane. Due to the increased visibility achieved with INT, network operators are able to troubleshoot problems more efficiently. Additionally, it is possible to perform instant processing in the data plane after measuring telemetry data (e.g., reroute flows when a link is congested), without having to interact with the control plane. Fig 20 shows an INT-enabled network. INT enables network administrators to determine the following:

- The path a packet took when traversing the network. Such information is difficult to learn using existing technologies when multi-path routing strategies (e.g., Equal-cost Multi-Path Routing (ECMP) [250], flowlet switching [251]) are used.
- The matched rules that forwarded the packets (e.g., ACL entry, routing lookup).
- The time a packet spent in the queue of each switch.
- The flows that shared the queue with a certain packet.

The P4 Applications Working Group developed the INT telemetry specifications [68] with contributions from key enablers of the P4 language such as Barefoot Networks, an Intel Company, VMware, Alibaba, and others.

INT allows instrumenting the metadata to be monitored without modifying the application layer. The metadata to be inserted depends on the use case; for example, if congestion was the main concern to monitor, the programmer inserts queue metadata and transit latency. An INT-enabled network has the following entities: 1) INT source: a trusted entity

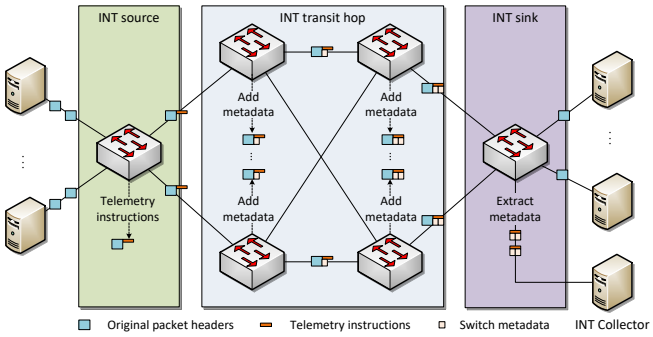


Fig. 20. In-band Network Telemetry (INT).

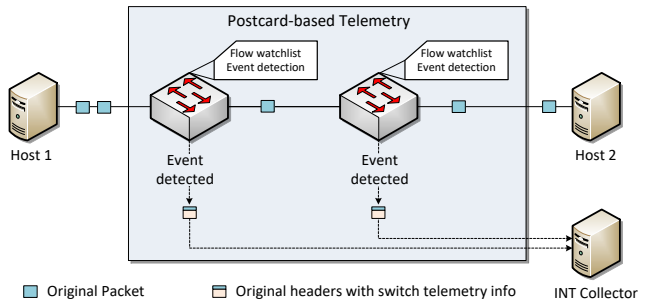


Fig. 21. Postcard-based Telemetry.

that instruments with the initial instruction set what metadata should be added into the packet by other INT-capable devices; 2) INT transit hop: a device adding its own metadata to an INT packet after examining the INT instructions inserted by the INT source; 3) INT sink: a trusted entity that extracts the INT headers in order to keep the INT operation transparent for upper-layer applications; and 4) INT collector: a device that receives and process INT packets.

The location of an INT header in the packet is intentionally not enforced in the specifications document. For example, it can be inserted as a payload on top of TCP, UDP, and NSH, as a Geneve option on top of Geneve, and as a VXLAN payload on top of VXLAN.

A. Postcard-based Telemetry (PBT)

INT provides the exact forwarding path, the timestamp and latency at each network node, and other information. Such detailed information is derived by augmenting user packets with data collected by each switch. PBT is an alternative to INT which does not modify user packets. Fig. 21 shows an example of PBT. As a user packet traverses the network, each switch generates a postcard and sends it to the monitor. The event that triggers the generation of the postcard is defined by the programmer, according to the application's need. Examples include start and/or end of a flow, sampling (e.g., one report per second), packet dropped by the switch, queue congestion, etc.

B. INT Variations

B.1. Background

Despite the improvements that INT brings compared to legacy monitoring schemes, it introduces bandwidth overhead when enabled unconditionally by network operators. In such scenarios, INT headers are added to *every* packet traversing the switch, increasing bandwidth overhead which decreases the overall network throughput. To mitigate such limitation, conditional statements are included in the P4 program to send reports only when certain events occur (e.g., queue utilization exceeds a threshold). This solution requires network operators to adjust thresholds and parameters manually based on the usual network traffic patterns. Consequently, several variations of INT have been developed, aiming at customizing its functionalities and addressing its limitations. Mainly, recent works focus on minimizing the bandwidth overhead of INT by adjusting thresholds and parameters automatically, based on measured traffic patterns and the desired application type.

B.2. Literature Review

Liu et al. [116] proposed NetVision, a telemetry system that aims at minimizing the traffic overhead of INT by way of probing. NetVision actively sends the rightful amount and format of probe packets depending on the telemetry application (e.g., traffic engineering, network visualization). To gain a comprehensive view of the network, the authors suggested customizing the probing path using Segment Routing (SR) [252]. The system was implemented on top of software switches (BMv2) organized in a fat-tree topology. The main limitation of this approach is that probing might result in poor accuracy and timeliness as the probes might experience different network conditions than actual packets.

Hyun et al. [117] proposed an architecture for self-driving networks that uses INT to collect packet-level network telemetry, and Knowledge-Defined Networking (KDN) to create intelligence to the network management, considering the collected telemetry data. KDN gets network information as input and generates policies to improve the network performance. The system was implemented on top of software switches (BMv2) connected to an ONOS controller.

Kim et al. [118] proposed selective INT (sINT), a scheme that dynamically adjusts the insertion frequency of INT headers. A monitoring engine observes changes in consecutive INT metadata and applies a heuristic algorithm to compute the insertion ratio. The goal of this work is to mitigate the bandwidth and network overhead issues of INT.

Marques et al. [119] described the orchestration problem in INT, which is associated with the optimal use of network resources for collecting the state and behavior of forwarding devices through INT. Instead of applying INT indiscriminately on all network traffic, this paper studies how telemetry tasks can use only a portion of traffic to gain full monitoring coverage, while minimizing the overhead. The problem was mathematically formalized and proven to be NP-Complete. Moreover, the authors proposed a heuristic algorithm to produce a polynomial-time solution to the problem.

Niu at al. [120] proposed multilayer INT (ML-INT), a system that visualizes IP-over-optical networks in realtime. The proposed system encodes INT headers in a subset of packets pertaining to an IP flow. The encoded headers contain

TABLE VIII
IN-BAND, POSTCARD-BASED, AND TRADITIONAL NETWORK TELEMETRY.

Feature	INT	PBT	Traditional
User packet modification	Yes	No	No
User packet overhead	Yes	No	No
Potential vulnerabilities	Higher	Lower	Lower
Flow tracking process	Simpler	More complex	More complex
Delay in reporting, tracking	Lowest	Low	High
Microbursts detection	Yes	Yes	No
Accuracy	Higher	Higher	Lower; especially with congested links
Reporting type	Push-based, initiated by the data plane	Push-based	Polling (e.g., SNMP), initiated by the control plane; sampling (e.g., NetFlow), initiated by the data plane
Troubleshoot problems	Easier and cheaper	Easier and cheaper	Harder and more expensive
Granularity	Higher; microseconds scale	Higher	Lower; milliseconds scale at best
Event-based monitoring	Customizable based on conditions and thresholds	Customizable	Not possible
Reactive processing	Faster; reactive processing is executed in the data plane	Faster	Slower; reactive processing is executed in the control plane
Bandwidth overhead	High when all packets are reported, low when reported based on events	Higher than INT	Lowest

metadata that describes statistics of electrical and optical network elements on the flow's routing path. The data plane program can be deployed on a switching ASICs or servers equipped with SmartNICs. The authors tested the system with a hardware switch (Tofino) and a SmartNIC (Netronome NFP-4000) installed on a general purpose server.

Ben et al. [121] proposed Probabilistic INT (PINT), an approach that probabilistically adds telemetry information into a collection of packets to minimize the per-packet overhead associated with regular INT. PINT bounds the overhead to limits defined by the operator. The motivation behind this work is that the majority of applications that leverage INT (e.g., congestion control, fast reroute) only require approximations of the telemetry data and therefore, do not need to gather per-packet per-hop INT information. To avoid the excessive INT overhead, the authors considered three different aggregation operations: 1) per-packet aggregation, which performs aggregation functions such as max/min/sum/product on packet values; 2) static per-flow aggregation, which compute all values on the path that are fixed for a given (flow, switch) pair (e.g., tracing a flow path using the ID of the switch); and 3) dynamic per-flow aggregation, which computes for each switch the stream of values (e.g., median flow latency on a certain switch). The authors implemented the system on a hardware switch (Tofino) and evaluated the approach by considering three use cases: 1) congestion control; 2) latency measurements; and 3) path tracing. Results show that the performance of PINT is comparable to state-of-the-art solutions but with a drastically lower overhead.

B.3. INT, PBT, and traditional telemetry comparison

Table VIII compares INT, PBT, and traditional telemetry. INT has higher potential vulnerabilities than PBT, such as eavesdropping and tampering. Adding extra protective measures (e.g., encryption) is difficult on the fast data path. On the other hand, PBT packets tolerate additional processing to enhance security. The flow tracking process is simpler with INT than with PBT. The latter requires the collector to correlate the multiple postcards of a single flow packet passing through the network, to form the packet history at the monitor. This process also adds delay in reporting and tracking. Legacy schemes that rely on sampling and polling suffer from accuracy issues, especially when links are congested. INT on the other hand is push-based, has better accuracy, and is more granular (microseconds scale). Reports sent by an INT-capable device contains rich information (e.g., the path a packet took) that can aid in troubleshooting the network. Such visibility is minimal in legacy monitoring schemes. Programmable switches permit reporting telemetry after the occurrence of specific events (e.g., congestion). Moreover, they provide flexibility in programming reactive logic that executes promptly in the data plane. One drawback of INT is that it imposes bandwidth overhead if configured to report for every packet; however, when event-based reports are considered, the bandwidth overhead significantly decreases.

C. INT Collectors

C.1. Background

An INT collector is a component in the network that processes telemetry reports produced by INT devices. It parses and filters metrics from the collected reports, then optionally stores the results persistently into a database. Since a large number of reports is typically produced in INT, having a high-performance collector is essential to avoid missing important network events. To this end, a number of research works focus on developing and enhancing the performance of INT collectors.

C.2. Literature Review

IntMon [122] provides an ONOS-based collector application for INT reports. In IntMon, INT information is inserted on top of TCP/UDP as shim headers. Each IntMon switch is equipped with a customized parser for INT headers, a forwarding table, and an INT processing block which adds/removes INT data

and sends INT packets to ONOS controller. IntMon uses a web-based ONOS applications to control which flows to monitor and the specific metadata to collect. There are two main limitations in this system; first, it does not allow querying historical network information as data is not stored; and second, it has low processing rate as it is implemented as an ONOS application.

Another INT collector is the Prometheus INT exporter [123] which extracts information from every INT packet and pushes its metric to a gateway. A database server then periodically pulls information from the gateway. The main problems with Prometheus INT exporter are the increased overhead of sending the data for every INT packet to the gateway, and the potential loss of network events as the database only stores the latest data pulled from the gateway.

INTCollector [124] is a collector for INT that extracts *events*, which are important network information, from INT raw data. It uses in-kernel processing to further improve the performance. INTCollector has two processing flows; the *fast path*, which processes INT reports and needs to execute quickly, and the *normal path* which processes events sent from the fast path and stores information in the database. INTCollector uses the eXpress Data Path (XDP) [253] to accelerate the packet processing in the kernel space.

DeepInsight [125] is a proprietary solution provided by Barefoot Networks, an Intel Company, that leverages INT capabilities to provide services such as real-time anomaly detection, congestion analysis, packet-drop analysis, etc. It is composed of the DeepInsight Analytics software which has a modular architecture and runs on commodity servers, and Barefoot SPRINT data plane telemetry which consists of the P4 program (INT.p4) and has intelligent triggers. It also provides open northbound RESTful APIs to allow customers to integrate their third-party network management solutions. Another proprietary solution is BroadView Analytics used on Broadcom Trident 3 devices by Broadcom [254].

Fig. 22 demonstrates the CPU efficiency of three INT collectors (IntMon, Prometheus INT exporter, and INTCollector) [124]. IntMon has the worst throughput, and is 57 times slower than Prometheus INT. INTCollector on the other hand has the best throughput and is 27 times faster than Prometheus INT exporter.

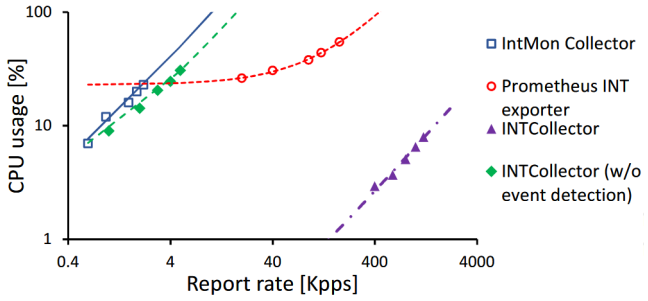


Fig. 22. CPU efficiency with the three INT collectors. Source: INTCollector paper [124].

C.3. Collectors in INT and legacy monitoring schemes comparison

Generally, collectors used with both INT and legacy monitoring schemes run on general purpose CPUs, and hence, have comparable performance. INT produces excessive amounts of reports when compared with legacy monitoring schemes (e.g., NetFlow), and therefore, requires having a collector with high processing capability. INT-based collectors are typically accelerated with in-kernel fast packet processing technologies (e.g., eXpress Data Path (XDP)) and hardware-based accelerators (e.g., Data Plane Development Kit (DPDK)).

D. Summary

Table IX summarizes the literature related to INT. Legacy telemetry tools and protocols are not capable of capturing *microbursts* nor providing network-wide packet visibility. INT was developed to address these challenges by enabling the data plane to query with high-precision the internal state of switches. Telemetry data are then embedded into packets and forwarded to a high-performance collector. The collector typically performs analysis and applies actions accordingly (e.g., informs the control plane to update table entries). Current research works focus on developing variations of the INT standard mainly to decrease the overhead of telemetry traffic. Other works focus on developing INT collectors that handle large volumes of traffic (in the scale of Kpps).

IX. Network Performance

Measuring and improving network performance is critical in nowadays' infrastructures. Low latency and high bandwidth are key requirements to operate modern applications that continuously generate enormous amounts of data [255]. Congestion control (CC), which aims at avoiding network overload, is critical to meet these requirements. Another important concept for expediting these applications is managing the queues that form in routers and switches through Active Queuing Management (AQM) algorithms. This section explores the literature related to measuring and improving the performance of programmable networks. This category can be further divided into the following sub-categories: 1) Congestion Control (CC); 2) measurements; 3) AQM; 4) Quality of Service (QoS) and Traffic Management (TM); and 5) multicast.

A. Congestion Control (CC)

A.1. Background

One of the most challenging tasks in the Internet today is congestion control and collapse avoidance [256]. The difficulty in controlling the congestion is increasing due to factors such as high-speed links, traffic diversity and burstiness, and buffer sizes [126]. Today's CC algorithms aim at shortening delays, maximizing throughput, and improving the fairness of network resources.

Tremendous amount of research work has been done on congestion control, including end hosts algorithms such as loss-based CC algorithms (e.g., CUBIC [257], Hamilton TCP (HTCP) [258], etc.), model-based algorithms (e.g., Bottleneck

TABLE IX
SUMMARY OF IN-BAND NETWORK TELEMETRY LITERATURE.

Category	Ref	Name	Description	Implementation notes
Variations	[116]	NetVision	Minimizes the overhead of INT and fixes scalability and coverage issues	Mininet
	[117]	N/A	Architecture for self-driving network by using INT and KDN	Software (BMv2) w/ ONOS controller
	[118]	sINT	Ratio of packets to be monitored can be adjusted depending on changes in network	Software (BMv2)
	[119]	N/A	INT Orchestration (INTO) problem and optimal use of network resources	N/A
	[120]	ML-INT	Visualize IP-over-optical networks in real-time	ASIC (Tofino) and SmartNIC (NFP-4000)
	[121]	PINT	Probabilistic INT over a collection of packets to minimize overhead	ASIC (Tofino)
Collectors / Solutions	[122]	IntMon	ONOS-based application for INT reports Rate: 0.1 Kpps	ONOS-BMv2 subsystem (ONOS 1.6)
	[123]	Prometheus INT exporter	Pushes INT data to gateway and database server pulls from the gateway Rate: 6.4 Kpps	ONOS P4 Brigade project
	[124]	INTCollector	Extracts events from INT raw data (event detection mechanism) Rate: 154.8 Kpps	XDP for in-kernel processing
	[125]	DeepInsight	Proprietary solution. Rate: N/A	SPRINT data plane telemetry (INT.p4)

Bandwidth and Round-trip Time (BBR) [259]), congestion-signalling mechanisms (e.g., Explicit Congestion Notification (ECN) [260]), data-center specific schemes (e.g., TIMELY [261], Data Center Quantized Congestion Notification (DC-QCN) [262], Data Center TCP (DCTCP) [263], pFabric [264], Performance-oriented Congestion Control (PCC) [265], etc.), and application-specific schemes (e.g., QUIC [266]).

With the advent of programmable data plane switches, researchers are investigating new methods to provide network-assisted congestion feedback for end-hosts.

A.2. Literature Review

Handley et al. [126] proposed NDP, a novel protocol architecture for datacenters that aims at achieving low completion latency for short flows and high throughput for longer flows. NDP requires modifying the whole stack, including the switch behavior, routing, buffer requirements (small buffers), and the transport protocol. It avoids core network congestion by applying per-packet multipath load balancing, which comes at the cost of reordering. It also trims the payloads of packets, similar to what is done in Cut Payload (CP) [267], whenever the queues of the switches become saturated. Once the payload is trimmed, the headers are forwarded using high-priority queues, notifying the receiver that the packet was lost. Consequently, a Negative ACK (NACK) is generated by the receiver and sent to the sender through high-priority queues, so that a retransmission is sent before draining the low priority queue. NDP was implemented using P4 on the NetFPGA-SUME platform as a proof of concept to show that NDP processing is simple enough to be deployed in programmable switches.

Turkovic et al. [127] proposed a P4-based congestion avoidance method that harnesses the power of programmable switches to track queueing delays of latency-critical flows, and to quickly react to congestion by rerouting flows to backup paths. The solution was implemented on both a software switch (BMv2) and on hardware through general-purpose servers enhanced with smart NICs. The solution was compared against congestion-agnostic and probing-based congestion-avoidance approaches. Results show that the average and maximum delay and jitter were enhanced compared to the other approaches.

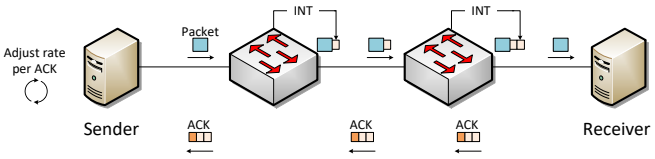


Fig. 23. HPCC: INT-based High Precision Congestion Control

Li et al. [128] proposed High Precision Congestion Control (HPCC), a new CC mechanism that leverages INT-based data added by P4 switches to obtain precise link load information. HPCC computes accurate flow rate by using only one rate update, as opposed to legacy approaches that require a large number of iterations to determine the rate. HPCC aims at enhancing the convergence time by using a Multiplicative-Increase Multiplicative-Decrease (MIMD) scheme. Note that previous TCP variants use the Additive-Increase Multiplicative-Decrease (AIMD), which is conservative when increasing the rate, and hence has a slow convergence time. The reason AIMD schemes are slow is that they use a single-bit congestion information (packet loss, ECN). With HPCC, end-hosts can perform aggressive increase as INT metadata encompasses precise link utilization and timely queue statistics. HPCC also provides near-zero queueing, while being almost parameterless. Fig. 23 shows the mechanism of HPCC. The switches add INT headers to every packet, and then the INT information is piggybacked into the TCP/RDMA Acknowledgement (ACK) packet. The end-hosts then use this information to adjust the sending rate through their smart Network Interface Controllers (NICs). The system was implemented on a hardware switch and on a commodity NIC with FPGA programmability in end-hosts.

Feldmann et al. [129] proposed a P4-based method that uses network-assisted congestion feedback (NCF) in the form of NACKs generated entirely in the data plane. NACKs are only sent to the elephant-flow senders in case of congestion, and hence is fast and efficient. The method maintains three separate

TABLE X
CONGESTION CONTROL SCHEMES. 1) ASSISTANCE FROM PROGRAMMABLE SWITCHES; 2) END-HOSTS; AND 3) LEGACY NETWORK-ASSISTED (ECN).

Characteristic	Programmable switch	End-hosts	Legacy network-assisted (ECN)
Accuracy	Higher, INT-based, microbursts are detected and reported	Low, packet loss (e.g., CUBIC); Medium, estimated RTT and btlbw (e.g., BBR)	Lower with classic ECN; High with L4S
Required modifications	Switches, end-hosts	None; distributed nature of AIMD does not require storing state of flows	Minimal if ECN is used (most equipment have classic ECN implemented); High if L4S is used
Convergence	Faster (MIMD)	Slower (AIMD)	Adequate with ECN; Fast with L4S ECN
Queue utilization	Near-zero	High; possibility of Bufferbloat (e.g., CUBIC)	Low
Parameterization	Few	None	Few (e.g., thresholds)
Congestion information	Several fields (e.g., queue occupancy, link utilization, flow share, etc.)	Packets drop	1-bit ECN mark

queues: mice flows queue, elephant flows queue, and control packets queue. Elephant flows are identified using rolling sketches without maintaining per-flow states. The authors claim that the approach works with both datacenters and Internet-wide scenarios. However, no implementation results were presented to evaluate the effectiveness of the solution.

Kfoury et al. [130] proposed a P4-based method to automate end-hosts' TCP pacing by supplying bottleneck bandwidths and large flows counts to senders in order to adjust their pacing rates to safe targets. Pacing decreases throughput variations and traffic burstiness, and hence minimizes queuing delays. This method is particularly applicable in networks where the number of large flows senders is small (e.g., science Demilitarized Zone (DMZ) [255]). The method was implemented on a software switch (BMv2), and results show improvements in fairness, throughput variations, and convergence time.

A.3. End-hosts, programmable switches, and legacy devices' CC schemes

Table X compares the CC schemes assisted by programmable switches (e.g., HPCC) with end-hosts CC algorithms (e.g., CUBIC) and legacy congestion signalling schemes (e.g., ECN). End-hosts CC infer congestion through packet drops and estimations (e.g., btlbw and Round-trip Time (RTT) estimation with BBR), which is not always sufficient to infer the existence of congestion. Legacy devices use classic ECN to signal congestion so that end-hosts slow down their transmission rates. Classic ECN is limited as it only marks a single bit to signal congestion, and is not aggressive nor immediate. Programmable switches on the other hand use fine-grained prompt measurements to signal congestion (e.g., INT metadata), which results in higher detection accuracy, near-zero queueing delays, and faster convergence time. The distributed nature of end-hosts CC schemes allows them to operate without modifying the network infrastructure and without tweaking parameters. ECN-enabled devices and programmable switches on the other hand require few parameters (e.g., marking threshold) to adapt to different network conditions.

B. Measurements

B.1. Background

Gaining an overall understanding of the network behavior is an increasingly complex task, especially when the size of the network is large and the bandwidth is high. Legacy measurements schemes have accuracy limitations since they rely on polling and sampling-based methods to gather traffic statistics. Typically, sampling methods have high sampling rates (e.g., one every 30,000 packets) and polling methods have large polling intervals. The literature [268] has shown that such methods are only suitable for coarse-grained visibility. The accuracy limitation of sampling and polling techniques hampers the development of measurement applications. For instance, it is not possible to accurately measure frequently changing TCP-specific fields such as congestion window, receive window, and sending rate.

Data streaming or sketching algorithms [269–273] were proposed to answer the limitation of sampling and polling. They address the following problem: *an algorithm is allowed to perform a constant number of passes over a data stream (input sequence of items) while using sub-linear space compared to the dataset and the dictionary sizes; desired statistical properties (e.g., median) on the data stream are then estimated by the algorithm*. The main problem with such algorithms is that they are tightly coupled to the metrics of interest. This means that switch vendors should build specialized algorithms, data structures, and hardware for specific monitoring tasks. With the constraints of CPU and memory in networking devices, it is challenging to support a wide spectrum of monitoring tasks that satisfy all customers. Legacy devices also lack the capability of customizing the processing behavior so that switches co-operate in the measurement process.

With the emergence of programmable switches, it is now possible to perform fine-grained measurements in the data plane at line rate. Moreover, data structures such as sketches and bloom filters can be easily implemented and customized for specific metrics of interest. Programmable switches paves the way for new areas of research in measurements since not only they provide flexibility in inspecting with high accuracy the traffic statistics, but also allow programmers to express reactive processing in real time (e.g., dropping a packet when

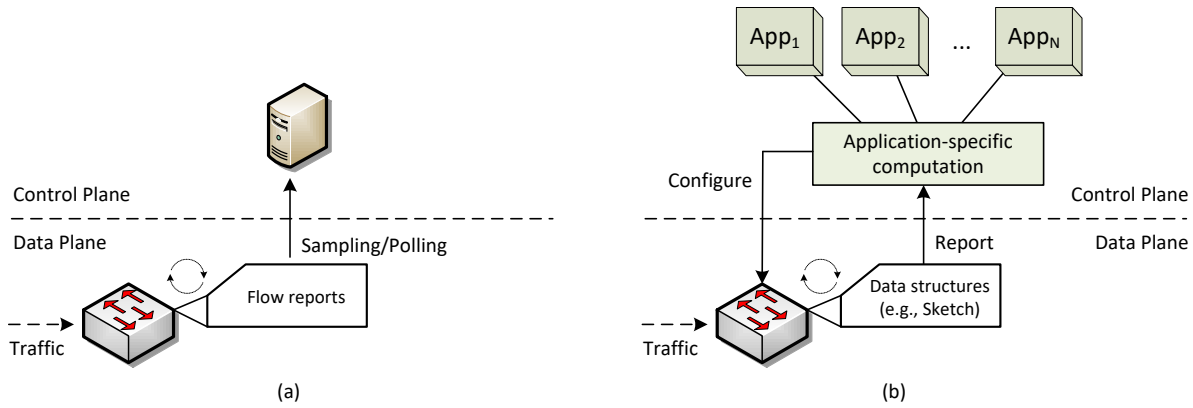


Fig. 24. (a) Traditional measurements with sampling/polling. (b) Measurements with programmable switches.

a threshold is bypassed as done in Random Early Detection (RED) [274]).

INT provides path-level metrics, with data similar to that of polling-based techniques; however, it is limited when the objective is to perform network-wide measurements. This section focuses on techniques that provide network-wide measurements. Fig. 24 shows the measurement schemes with legacy devices and with programmable switches.

B.2. Literature Review

Li et al. [131] proposed FlowRadar, a system that stores counters for all flows in the data plane with low memory footprint, then exports periodically (every 10ms) to a remote collector. The collector can then perform network-wide decoding and analysis of the flow counters. FlowRadar was designed as a solution to NetFlow's limitation of monitoring only subset of packets. The solution was implemented on a software switch (BMv2).

Liu et al. [132] proposed Universal Monitoring (UnivMon), a monitoring framework based on programmable switches that provides accuracy and generality across a wide range of monitoring tasks. The main motivation behind this work is that existing methods that are polling-based are inaccurate, and others streaming and sketching-based methods craft customized algorithms for specific application-level metrics, which is not scalable for the long term. UnivMon aims at providing an application-agnostic monitoring solution, while using the granularity of the data plane to improve accuracy. Different estimation algorithms run in the control plane, and use statistics collected by the data plane through sketches (see Fig. 24 (b)). The solution was implemented on top of a software switch (BMv2) and the results were compared against OpenSketch [275].

Narayana et al. [133] presented Marple, a query language for performance monitoring in the data plane. Marple is based on common query constructs (i.e., map, filter, group by), and uses key-value store primitives stored on hardware switches. Essentially, network operators express their need using queries, which are fed into the Marple compiler. The output is a P4 program to be deployed on the target switch. Marple allows performing advanced aggregation (e.g., moving average of latencies) at line rate and can scale to millions of keys. The switches stream the results to collecting servers for further processing. Marple was tested on a software switch (BMv2), and the authors evaluated the hardware resources and the memory evictions tradeoffs for the key-value store.

Ghasemi et al. [134] proposed Dapper, an in-network TCP performance diagnosis system. Dapper analyzes packets in real time, and identifies and pinpoints the root cause of the bottleneck (sender, network, or receiver). TCP state, congestion and receive windows are inferred after examining packet headers, sizes, timing, and other metadata. Dapper infers sending rate, Maximum Segment Size (MSS), sender's reaction time (time between received ACK and new transmission), loss rate, latency, congestion window (CWND), receiver window (RWND), and delayed ACKs. The system was implemented on hardware switch (Tofino) and the emulation used real-world packet traces from the Center for Applied Internet Data Analysis (CAIDA).

Yang et al. [135] proposed Elastic sketch, a network measurement solution that is adaptive to changes in network conditions (e.g., bandwidth, packet rate and flow size distribution). Elastic sketch first separates elephant flows from mice flows, then performs sketch compression in order to improve scalability. Elastic sketch is generic, and can work for a variety of measurement tasks, namely, flow size estimation, heavy hitter/change detection, flow size distribution estimation, entropy estimation, cardinality estimation, and others. The system was implemented on a hardware switch (Tofino).

Yaseen et al. [136] proposed SpeedLight, a system that performs network-wide measurements using programmable switches. The legacy systems which use counter polling and packet sampling are limited to the visibility of the individual entity in the network. The proposed system records synchronized traffic statistics by capturing local measurements that together provide a coherent image of the entire network. The solution was implemented on a hardware switch (Tofino) and the authors evaluated the synchronicity of the flows and the scalability of the solution. Furthermore, a load balancing scheme that uses SpeedLight was described and evaluated as a use case.

Joshi et al. [137] proposed BurstRadar, a system that uses programmable switches to monitor microbursts entirely in the

data plane. Mircorbursts are events of sporadic congestion that last for tens or hundreds of microseconds. Microbursts increase latency, jitter, and packet loss, especially when links' speeds are high and switch buffers are small. It is almost impossible to detect microbursts in legacy switches which use sampling and polling-based techniques. BurstRadar detects microbursts, and captures a snapshot of the telemetry information of all the involved packets. Afterwards, an analysis is conducted on the snapshot to identify the microburst-contributing flow and the burst characteristics. The system was implemented on a hardware switch (Tofino) and the evaluations used microbursts traffic distribution from Facebook's production network. Results showed that BurstRadar improved the processing overhead and the amount of data collected by ten times compared to existing approaches.

Sonchack et al. [139] presented *Flow, a telemetry measurement system that supports concurrent measurements and dynamic queries. The main motivation behind this work is that compiling excessive queries into the data plane can result in concurrency problems and network disruption. A query can be divided into three operations: 1) *select*, which determines the packet headers and metadata to be captured; 2) *group*, which describes how to attribute packets to a certain flow (e.g., 5-tuple); and 3) *aggregate*, which define the statistics to be computed over the packets' features. *Flow leverages programmable switches to perform the *select* and *group* operations as they can be easily implemented in the data plane at line rate. The *aggregate* operation however requires more computations and is very limited if performed in the data plane. Therefore, the authors delegated this operation to commodity servers which can perform coarser grained grouping and mathematical computation very efficiently. The solution was implemented on a hardware switch (Tofino) to demonstrate its practicality and to show that different applications can be scaled to support terabit rates measurements.

Sonchack et al. [140] proposed TurboFlow, a flow record generator based on programmable switches that operates entirely in the data plane without any support from external servers. The data plane generates microflow records, which are the most recent packets in a flow, aiming at reducing the amount of required state storage. The state is stored in a hash table-like data structure that replaces the old microflow upon collision with the new one, while sending the former to the switch's CPU. Afterwards, the microflows are sent to the switch's memory through Direct Memory Access (DMA), and are then grouped into full flow records. TurboFlow was evaluated on a hardware switch (Tofino) and the results show that it is scalable and can operate with both Internet and data center workloads.

Gupta et al. [141] proposed Sonata, a query-driven network telemetry system that uses both programmable data plane switches and scalable stream processors. It provides a unified query interface that allows network operators to use common data flow operators (e.g., filter, reduce, etc.) over packet header fields. Sonata computes optimal query plans through Integer Linear Programming (ILP) to minimize traffic sent from the data plane to the stream processor. The system was implemented on top of both hardware (Tofino) and software (BMv2) switches.

Lee et al. [138] designed an in-network system that passively measures latency and RTT so that service level agreements (SLA) are monitored. SLAs are legally binding contracts that describes the expected QoS metrics. Existing SLA verification methods use active measurements, which give a poor representation of the traffic. The proposed system stores the timestamps of outgoing packets and their corresponding ACKs to compute the RTT.

Chen et al. [142] proposed ConQuest, a P4-based queue measurement solution that determines the size of flows occupying the queue in real time, and identifies flows that are grabbing a significant portion of the queue. This solution is useful for use cases such as mitigating congestion-based attacks, avoiding conflicting workloads, implementing new AQMs, optimizing switch configurations, debugging switch implementation, and off-path monitoring of queues in legacy devices. ConQuest is based on a data structure that allows identifying flows depending on the purpose (e.g., detecting bursty connections). It maintains compact snapshots of the queue, updated on each incoming packet. The snapshots are then aggregated in a round-robin fashion to approximate the queue occupancy. Afterwards, it cleans the previous snapshots to reuse it for further packets. The authors implemented ConQuest on a hardware switch (Tofino) and showed that the solution is feasible in terms of resource consumption such as the required pipeline stages, the number of registers, and the complexity of actions.

Liu et al. [143] proposed a memory-efficient approach for network performance monitoring. The authors used lean algorithms that use memory sublinear in respect to the number of flows and the size of the input data. The solution allows monitoring new metrics such as retransmissions, packet loss, round-trip-time, out-of-order packets, and others. The system was implemented on a hardware switch (Tofino), and results show that only 40KB of memory was used to detect 82% of top 100 problematic flows found in real-world packet traces.

Holterbach et al. [144] proposed Blink, a framework that detects failures in the data plane and promptly reroutes after signals generated by the data plane. The key idea is that TCP's behavior is predictable upon encountering failure, where the same packet is retransmitted at epochs exponentially spaced in time. Blink operates entirely in the data plane, and is able to restore connectivity in sub-second. The system was developed on a hardware switch (Tofino), and the emulation considered traces from real-world captures (CAIDA).

Ding et al. [145] proposed P4Entropy, an algorithm to estimate network traffic entropy (Shannon entropy) in the data plane. Tracking entropy is useful for calculating traffic distribution in order to understand the network behavior. The system was implemented on a software switch (BMv2).

Wang et al. [146] proposed SpiderMon, a system that performs network-wide diagnosis of performance degradation. The idea is to have every switch maintain fine-grained telemetry data for a short period of time, and upon detecting performance degradation (e.g., increased delay), the information is offloaded to collector. The collector then performs analysis on the received metadata from several switches to derive the

cause of the performance problem. The systems imposes low overhead as only a fraction of the telemetry data is sent to the collector. The system is implemented on a software switch (BMv2).

Teixeira et al. [147] proposed PacketScope, a system that allows network operators to query the internal switch processing (e.g., headers modification, packet drop, delay, etc.) through dataflow constructs. PacketScope enables monitoring on the ingress and the egress pipelines of the switch, a feature that was not developed in previous query-driven approaches like Sonata and Marple. The system was implemented on a software switch (BMv2) as an extension to Sonata.

Bai et al. [148] proposed FastFE, a system that performs traffic features extraction by leveraging programmable data planes. Features extraction is particularly important for traffic analysis and behavior detector ML techniques. Contemporary traffic analysis methods face severe scalability and overhead issues when analyzing high-speed large-volume traffic. In FastFE, operators are given an interface that enables them to express the traffic features they desire as policies. A policy enforcement engine in the system maps those policies into switch and server programs. The server programs are generated in case some primitives cannot be implemented on the switch (e.g., insufficient resources, executing complex computations). Note that FastFE tries to fully utilize the programmable switch whenever it is possible. The system was implemented on a hardware switch (Tofino) and was tested with Kistune [276], a neural network based network Intrusion Detection System (IDS). Results show that FastFE produces similar feature vectors to Kistune, improves scalability, and has low resource overhead on the switch.

Chen et al. [149] implemented a data plane program that measures the RTT of TCP traffic in ISP networks. RTT measurement is important for detecting spoofing and routing attacks, ensuring SLAs compliance, measuring the Quality of Experience (QoE), improving congestion control, and many others. RTT is measured in today's networks using active measurement tools (e.g., PingMesh [277], Network Diagnostic Tool (NDT) [278], perfSONAR [279]) through probes. However, ISPs cannot utilize such tools since they do not possess control over end-hosts to run agents on them. On the other hand, passive measurement tools (e.g., Ruru [280]) are not capable of capturing RTTs over the lifetime of a TCP session. The system proposed in this paper continuously measures the RTT of all packets using programmable switches placed in the ISP network. The program measures the RTT by matching the sequence number of an outgoing packet with the acknowledgment number of an incoming packet. The authors leveraged the stateful processing of programmable switches to store the record of packets in the memory. The system was implemented on a hardware switch (Tofino) and the results show that 99% of the RTT samples of a 10Gbps link were collected.

Qiu et al. [150] implemented time-adaptive sketches on programmable data plane. The motivation of this work is that recently captured traffic trends are the most relevant in network monitoring and attacks'detection. The notion of time is not supported by native streaming algorithms. For instance, *count-min sketch*, which is a data structure that uses constant memory amount to record data, is oblivious to the passage of time. Existing solutions that consider recency are easily implemented on software, but not on programmable ASICs. For example, resetting a sketch after a timer expires requires iterating over the elements in the sketch, an operation that cannot be implemented in the data plane due to the lack of loops. Likewise, creating multiple sketches require additional stages which is limited in the hardware. Time-adaptive sketches utilize the idea of Dolby noise reduction [281, 282]; a *pre-emphasis* function inflates the update when a new key is inserted and a *de-emphasis* function restores the original value. This mechanism ages the old events over time, and therefore, improves the accuracy of recent events. The authors implemented the pre-emphasis function in the data plane using simple bit shifts, and the de-emphasis function in the control plane. The system was implemented on a hardware switch (Tofino), and the results show that adding timing in the sketch requires additional 1-2 stages and a negligible increase in the SRAM.

Huang et al. [151] proposed OmniMon, an architectural design that coordinates flow-level network telemetry operations between programmable switches, end-hosts, and controllers. Such coordination aims at achieving high accuracy while maintaining low resource overhead. OmniMon follows a *split-merge* strategy where the *split* operation decomposes telemetry operations into partial operations and schedules them among the entities (switches, end-hosts, and controller), and the *merge* operation coordinates the collaboration among these entities. The idea is to leverage the strength of the data plane in the switches and end-hosts (i.e., per-flow measurements with high accuracy) and the control plane (i.e., network-wide collaboration). OmniFlow also ensures *consistency* through a synchronization mechanism and *accountability* through a system of linear equation considering packet loss and other data center characteristics. The system was implemented on a hardware switch (Tofino) and on commodity servers (DPDK). Results show that OmniMon reduces the memory by 33%-96% and the number of actions by 66%-90% when compared to state-of-the-art solutions.

Chen et al. [152] proposed BeauCoup, a P4-based measurement system that handles multiple heterogeneous queries in the data plane. It offers a general query abstraction that counts the *attributes* across related packets identified by *keys*, and flags packets that surpass a defined threshold. For example, to detect worms, an operator might use the source IP as the *key* and the destination IP as the *attribute*. BeauCoup is capable of executing multiple concurrent queries while considering the resource limitations of the switch. Moreover, operators have the flexibility of modifying the queries at runtime without recompiling the data plane program. The design of BeauCoup follows the coupon-collection problem [283], which computes the number of random draws from n coupons such that all coupons are drawn at least once. For example, if the threshold of distinct destination IPs for detecting superspreaders is 130, instead of recording all distinct destination IPs, 32 coupons are defined. Consequently, the destination IPs of incoming packets are mapped to those 32 coupons. BeauCoup was implemented

on a hardware switch (Tofino), are results show that it uses 4× less memory than state-of-the-art measurement sketches.

B.3. In-network versus legacy measurements

There are two main classes of legacy measurements techniques. First, there are techniques that rely on polling and sampling (e.g., NetFlow). The differences between in-network measurements and polling/sampling-based schemes are closely related to those described in Table VIII. Second, there are techniques that rely on sketching or streaming algorithms to estimate the metric of interest. Such methods are tightly coupled with the metric, which forces hardware vendors to invest time and effort in building customized algorithms and data structures that might not be used by various customers. Moreover, with the constraints of routers and switches, it is not possible to implement a variety of monitoring tasks while still supporting the standard routing/switching functionalities. Therefore, such approaches are not scalable for the long run. With programmable switches, it is possible to customize the monitoring tasks by implementing customized sketching/streaming algorithms as P4 programs. This advantage improves scalability as the operator can always modify the algorithms whenever needed.

C. Active Queue Management (AQM)

C.1. Background

A fundamental component in network devices is the *queue* which temporarily buffers packets. As data traffic is inherently bursty, routers have been provisioned with large queues to absorb this burstiness and to maintain high link utilization. The majority of delays encountered in a communication session is a result of large backlogs formed in queues. Previous legacy devices are limited in the visibility of the queue as they provide little or no insight about which flows are occupying or sharing the queue [142]. Consequently, researchers have been investigating queue management algorithms to shorten the delay and mitigate packet losses, while providing fairness among flows.

AQM is a set of algorithms designed to shorten the queueing delay by prohibiting buffers on devices from becoming full. The undesirable latency that results from a device buffering too much data is known as "Bufferbloat". Bufferbloat not only increases the end-to-end delay, but also decreases the throughput and increases the jitter of a communication session. Modern AQMs help in mitigating the bufferbloat problem [284–287]. Unfortunately, modern AQMs are typically not available in state-of-the-art network equipment; for instance, Controlled Delay (CoDel) AQM, which was proposed in 2013, and was proven in the literature to be effective in mitigating Bufferbloat [288], is still not available in most network equipment. With programmable switches, it is now possible to implement AQMs as P4 programs, which not only accelerates support for new AQMs, but also provides means to customize its parameters programmatically in response to network traffic. Moreover, programmable switches thrives for innovation on newer AQMs that can be easily implemented and rapidly tested.

C.2. Literature Review

Kundel et al. [153] implemented CoDel queueing discipline on a programmable switch. CoDel eliminates Bufferbloat, even in the presence of large buffers [289], also known as bloated buffers. Its algorithm can be easily expressed in the data plane as it consists of comparisons, counting, basic arithmetic, and dropping packets. The implementation was realized on a software switch (BMv2), and the conducted evaluations confirm its expected behavior.

Sharma et al. [154] proposed Approximate Fair Queueing (AFQ), a mechanism built on top of programmable switches that approximates fair queuing on line rate. Fair Queueing (FQ) aims at fairly dividing the bandwidth allocation among active flows. FQ algorithms in general are difficult to implement on hardware as they require complex flow classification, per-packet scheduling, and buffer allocation. Such requirements make FQ algorithms expensive to be implemented on high-speed devices. Consequently, AFQ aims at *approximating* fair queueing by using programmable switches' features such as mutating switch state, performing basic calculations, and selecting the egress queue of a packet. AFQ's operations can be summarized as follows: 1) per-flow state, which includes the number and timing information of the previous packet pertaining to that flow, is approximated; 2) the position of each packet in the output schedule is determined; 3) the egress queue to use is selected; and 4) the packet is dequeued based on the approximate sorted order. The solution was implemented on a hardware switch (Cavium) and metrics such as Flow Completion Time (FCT), normalized incast latency, average bytes and retransmissions dropped per flow, are reported. The results show that AFQ presents improved performance over the existing schemes, and has fairly minimal overhead regarding hardware resources.

Laki et al. [155] described an AQM evaluation testbed with P4 in a demo paper. The authors tested the framework with two AQMs: Proportional Integral Controller Enhanced (PIE) and RED. Mushtaq et al. [290] approximated Shortest Remaining Processing Time (SRPT), and prototyped their solution on the software switch (BMv2).

Papagianni et al. [156] implemented Proportional Integral PI^2 AQM on a programmable switch. PI^2 is an extension of PIE AQM to support coexistence between classic and scalable congestion controls in the public Internet. PI^2 is simple to implement as it is mostly based on basic bit manipulations. The system was implemented on a software switch (BMv2), and was verified by comparing its results with the expected results from the original PI^2 paper [291].

C.3. AQMs on programmable switches and fixed-function devices

Inventing novel AQMs that control queueing delay, mitigate bufferbloat, and achieve fairness with different network conditions (e.g., short/long RTTs, lossy networks, WANs) is an active research area. Typically, new AQMs are implemented and tested in software (e.g., as a Linux queueing discipline (qdisc) used with traffic control (tc)), which is limited for evaluating novel AQMs on production networks. With programmable switches, AQMs are implemented in P4 programs,

TABLE XI
AQMs ON PROGRAMMABLE AND FIXED-FUNCTION SWITCHES.

Feature	Programmable switches	Fixed-function devices
Innovation	Higher; new AQMs are expressed in P4 programs	Lower; only developed by equipment vendors
Exclusivity	Higher; operators can implement their own custom AQMs without disclosing technical information	Lower; most supported AQMs are standards
Readiness	Faster (weeks to months); once an AQM is expressed in P4, it can be immediately available	Slower (years)
Cost	Lower	Higher
Tweakable	Higher; even standard AQMs can be customized and tweaked based on network traffic	Lower; only through parameters

which foster innovation and enhance testing with production networks. Additionally, operators can create their own customized AQMs that perform efficiently with their typical network traffic. Historically, deploying AQMs on network devices is a lengthy and costly process; once an effective AQM is published and thoroughly tested, equipment vendors start investigating whether it is feasible to implement it on future devices. Such process might take years to finish, and by then, new network conditions evolve, requiring new AQMs. With programmable switches, this process is cost-efficient and relatively fast (can be completed in weeks). Table XI compares the features of AQMs on programmable switches versus fixed-function devices.

D. Quality of Service and Traffic Management

D.1. Background

Meeting diverse Quality of Service (QoS) requirements is a fundamental challenge in today's networks. Traffic Management (TM) provides access control that guarantee that the traffic admitted to the network conforms to the defined QoS guarantees. TM often regulates the rate of a flow by applying traffic policing. New generation of programmable switches facilitate traffic policing and differentiation by allowing network operators to express their logic in a programming language (P4). This section explores the works on programmable switches that involve QoS and TM.

D.2. Literature Review

Bhat et al. [157] described a system where programmable switches route traffic intelligently by inspecting application headers (layer-5) to improve users' QoE. The idea is to translate application-layer header information into link-layer headers (Q-in-Q 802.1ad) for the core network in order to perform QoS routing and provisioning. The authors adopted the Adaptive Bit Rate (ABR) video streaming as a use case to showcase the QoS improvements and the flexibility of traffic management. The system is implemented on a software switch (BMv2), and was tested on an actual research testbed (Chameleon).

Lee et al. [158] implemented a traffic meter based on Multi-Color Markers (MCM) on programmable switches to support multi-tenancy environments. A Virtual Network (VN) has to have its own dedicated bandwidth (i.e., other networks' traffic should not impact the bandwidth) and should be able to differentiate priorities in order to provide QoS for its flows. The system was implemented on a software switch (BMv2).

Tokmakov et al. [159] proposed RL-SP-DRR, a traffic management system that combines Rate-limited Strict Priority (RL-SP) and Deficit round-robin (DRR) to achieve low latency and fair scheduling while improving link utilisation, prioritization and scalability. The system was implemented on a software switch (BMv2) and was evaluated by comparison against standard priority-based and best-effort scheduling.

Chen et al. [160] proposed a bandwidth manager for end-to-end QoS provisioning using programmable switches. The system classifies packets into different categories based on their QoS demands and usages, and uses two-level queue when prioritizing. The system aims at limiting the maximum allowed rate and at maximizing bandwidth utilization. Thea authors implemented the idea on a hardware switch (Tofino) and the design is compared against approaches based on OpenFlow.

D.3. Comparison of QoS/TM between legacy and programmable networks

The ability to perform QoS-based traffic management in legacy networks is restricted to algorithms that consider standard header fields (e.g, differentiated services [292]). On the other hand, programmable switches can parse, modify, and process customized protocols. Hence, operators now have the ability to perform TM by inspecting custom headers fields. Moreover, it is possible to extract with high-granularity metadata pertaining to the state of the switch (e.g., queue occupancy, packet sojourn time, etc.) at line rate. Such information can significantly help switches take better decisions while performing traffic management.

E. Multicast

E.1. Background

Multicast routing enables a source node to send a copy of a packet to a group of nodes. Multicast uses in-network traffic replication to ensure that at most a single copy of a packet traverses each link of the multicast tree. Perhaps the most widely multicast routing protocol deployed in traditional networks is the Protocol-Independent Multicast (PIM) protocol [293]. PIM and other multicast routing protocols require a signaling protocol such as the Internet Group Management Protocol (IGMP) [294] to create, change, and tear-down the multicast tree. Traditional multicast presents some challenges. For example, it is not suitable for environments where multicast group members constantly move (e.g., virtual machine migration and allocation). In such cases, the multicast tree must be updated dynamically, which may require substantial time and overhead. Also, some routers support a limited number of group-table entries, which does not scale in environments such as datacenters. Additionally, the signaling protocol and multicast algorithm are hard coded in the router, which reduces

TABLE XII
COMPARISON BETWEEN P4-BASED AND TRADITIONAL MULTICAST.

Feature	P4-based multicast	Traditional multicast
Scalability	High; no state information required in switches	Low; state information required in switchers per-group
Tree management	Flexible; custom multicast algorithm and features can be implemented	Inflexible; signaling protocol required and hard coded in the switch
Packet overhead	High; multicast tree is encoded in packet header	No packet overhead
Dynamic tree updates	Easy; packet header carries update information	Complex; topology challenges may trigger time-consuming tree changes
IP address constraint	Flexible; switch can multicast packets independently of the type of IP address	Fixed; switch is hard-coded to only multicast packets with destination IP address in the range 224.0.0.0 - 239.255.255.255

flexibility in building and managing the tree. Finally, it is not possible to implement multicast based on non-standard header fields.

E.2. Literature Review

Shahbaz et al. [161] presented ELMO, a multicast scheme based on programmable P4 switches for datacenter applications. ELMO encodes the multicast tree in the packet header, as opposed to maintaining group-table entries inside routers. Kadosh et al. [162] implemented ELMO using a hybrid dataplane with programmable and non-programmable elements. ELMOS is intended for multi-tenant datacenter applications requiring high scalability. Braun et al. [163] presented an implementation of the Bit Index Explicit Replication (BIER) architecture [295] with extensions for traffic engineering. Similar to ELMO, BIER removes the per-multicast group state information from switches by adding a BIER header, which is used to forward packets. BIER does not require a signaling protocol for building, managing, and tearing down trees.

E.3. Comparison P4-based and traditional multicast

Table XII compares P4-based multicast and traditional multicast. The main advantages of implementing multicast routing with programmable P4 switches are: i) the group membership is encoded in the packet itself, which permits the creation of arbitrary multicast tree based on the application. For example, a multicast tree to update software devices may prioritize bandwidth over latency, while one for media traffic may prioritize latency; ii) switches do not need to store per-group state information, although tables can be customized and used in conjunction with the tree encoded in the packet header; iii) groups can be reconfigured easily by changing the information in the header of the packet; and iv) the elimination of the signaling protocol to build, manage, and tear-down the tree results in consider simplification and flexibility for the operator.

F. Summary

Table XIII summarizes the works pertaining to network performance. Performing network-wide monitoring and measurements is of utmost importance for network operators to debug performance degradation. There is a wide range of works on programmable switches to improve the accuracy of the measurements as well as for providing a broader view of the network. Another line of research is to mitigate congestion and losses by analyzing measurements collected in the data plane and by applying queue management policies. Furthermore, a handful of works are investigating methods to improve QoS by applying traffic policing and management. Finally, the scalability concerns of multicast in legacy networks are being mitigated with programmable switches.

X. MIDDLEBOX FUNCTIONS

RFC 3234 [296] defines *middlebox* as a device that performs functions other than the standard functions of an IP router between a source and a destination host. In legacy devices, middlebox functions are designed and implemented by manufacturers. Hence, they are limited in the functionalities they provide, and typically include standard well-known functions (e.g., NAT, protocol converters (6to4/4to6), etc.). To overcome this limitation, the trend moved towards implementing middleboxes in x86-based servers and in data centers as Network Function Virtualization (NFVs). While this shift accelerated innovation and introduced a wide range of new applications, there was some performance implications resulting from operating systems' scheduling delays, interrupt processing latency, pre-emptions, and other low-level OS functions. Since programmable switches offer the flexibility of inspecting and modifying packets' headers based on custom logic, they are excellent candidates for enabling middlebox functions, while operating at line rate without performance implications.

A. Load Balancing

A.1. Background

A cloud data center, such as a Google or Facebook data center, provides many applications concurrently, such as email and video applications. To support requests from external clients, each application is associated with a publicly visible IP address to which clients send their requests and from which they receive responses. This IP address is referred to as Virtual IP (VIP) address. The external requests are then directed to a software load balancer whose task is to distribute requests to the servers, balancing the load across them. The load balancer is also referred to as layer-4 load balancer because it makes decisions based on the 5-tuple source IP address and port, destination IP address and port, and transport-layer protocol. This state information is stored in a connection table containing the 5-tuple and the Direct IP (DIP) address of the server serving that connection. State information is needed to avoid disruptions caused by changes in the DIP pool (e.g., server failures, addition of new servers). The load balancer also provides a translation functionality, translating the VIP to the internal DIP, and then translating back for packets traveling

TABLE XIII
SUMMARY OF NETWORK PERFORMANCE LITERATURE

Category	Ref	Name	Description	Implementation		
				Hardware (Tofino)	Software (BMv2)	Others
Congestion Control	[126]	NDP	Data center architecture to achieve low latency and high throughput			✓
	[127]	N/A	Rerouting flows to backup paths		✓	
	[128]	HPCC	Compute accurate flow rate from INT-based data	✓		
	[129]	NCF	Network-assisted congestion feedback sent to elephant flows			✓
	[130]	N/A	Supply network information to end-hosts to apply pacing		✓	
Measurements	[131]	FlowRadar	Stores and periodically exports statistics to remote collector		✓	
	[132]	UnivMon	Application-agnostic monitoring solution		✓	
	[133]	Marple	Query language for performance monitoring in the data plane		✓	
	[134]	Dapper	In-network TCP performance diagnosis system	✓		
	[135]	Elastic	Measurement solution adaptive to network changing conditions	✓		
	[136]	SpeedLight	Synchronized traffic statistics that together provide a coherent image of the entire network	✓		
	[137]	BurstRadar	Monitor microbursts entirely in the data plane	✓		
	[139]	*Flow	Concurrent measurements and dynamic queries	✓		
	[140]	TurboFlow	Flow record generator without any support from external servers	✓		
	[141]	Sonata	Query-driven network telemetry system that uses switches and scalable stream processor	✓		
	[142]	ConQuest	Queue measurements inside the data plane	✓		
	[143]	N/A	Memory-efficient approach for performance monitoring using lean algorithms	✓		
	[144]	Blink	Detects failures in the data plane and promptly reroutes traffic	✓		
	[145]	P4Entropy	Estimate network traffic entropy in the data plane		✓	
	[146]	SpiderMon	Network-wide diagnosis of performance degradation		✓	
	[147]	PacketScope	Querying switch processing through dataflow constructs		✓	
	[148]	FastFE	Traffic feature extraction by leveraging data plane	✓		
	[149]	N/A	Measures the RTT of TCP traffic in ISP networks	✓		
	[150]	N/A	Time-adaptive sketches on programmable data plane	✓		
	[151]	OmniMon	Architectural design that coordinates flow-level telemetry	✓		
	[152]	BeauCoup	Measurement that handles multiple heterogeneous queries	✓		
AQM	[153]	P4-CoDel	Implementation of CoDel on P4		✓	
	[154]	AFQ	Approximate fair queueing in the switch			✓
	[155]	N/A	Evaluation testbed for PIE and RED		✓	
	[156]	N/A	Implementation of PI^2 on P4		✓	
QoS and TM	[157]	N/A	Routing traffic by inspecting application headers		✓	
	[158]	N/A	Traffic meter based on multi-color markers		✓	
	[159]	RL-SP-DRR	Traffic management that combines RL-SP and DRR		✓	
	[160]	N/A	Bandwidth manager for end-to-end QoS provisioning	✓		
Multicast	[161]	ELMO	Encodes the multicast tree in the packet header	✓		
	[162]	ELMO	Implementation of elmo on hybrid dataplane			✓
	[163]	N/A	Adds BIER header, which is used to forward packets		✓	

in the reverse direction back to the clients. The traditional software-based load balancer is illustrated in Fig. 25(a).

A.2. Literature Review

Recent works presented schemes where load balancing functionality is implemented in programmable P4 switches. The main idea consists of storing state information directly in the switch's dataplane. The connection table is managed by the software load balancer, which can be implemented either in the switch's control plane or as an external device, as shown in Fig. 25(b). The software load balancer adds new entries in the switch's table as they arrive, or removes old entries as flows end.

Katta et al. [164] proposed HULA, a load balancer implemented in the data plane. It basically solves the problems of the canonical load balancing mechanism, ECMP, and its successor, CONGA [297], which is implemented on custom hardware. Instead of storing the congestion status of all paths in leaf switches, a HULA-enabled switch stores the best path to the destination via its neighboring switch. The authors implemented HULA as a P4 program, but did not provide details on the target type. HULA was also implemented on ns-2 discrete event simulator and compared against ECMP and CONGA.

Miao et al. [165] proposed SilkRoad, a load balancer that provides a direct path between application traffic and servers. It eliminates the need for a software-layer while mapping a connection to the same server, ensuring Per-Connection Consistency (PCC) property. The main advantage of switch-based load balancer is the throughput. By offloading the logic to the ASIC, SilkRoad can load balance ten million connections simultaneously, an order of magnitude or more than that supported by software load balancers. The system was implemented on a hardware switch (Tofino), and results show that a single switching device is capable of replacing up to hundreds of software load balancers.

In the presence of multi-path transport protocols (e.g., Multi-path TCP (MPTCP) [298]), systems such as CONGA and HULA provide sub-optimal forwarding decisions when

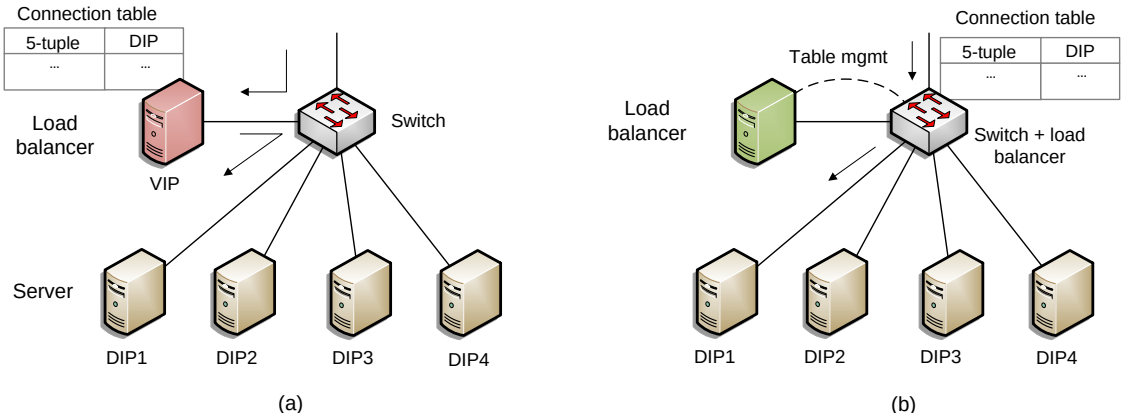


Fig. 25. (a) Traditional software-based load balancing. (b) Load balancing system implemented by a programmable switch.

several subflows pertaining to a single connection are pinned on the same bottleneck link. As a result, Benet et al. [166] proposed MP-HULA, a transport layer multi-path aware load-balancing scheme that uses the best-k paths to the destination through the neighbor switches. MP-HULA was evaluated on ns2 and compared against ECMP and CONGA, but was not implemented on a hardware switch.

Olteanu et al. [167] proposed Beamer, a data center load balancer that is stateless and scalable, and can support both TCP and MPTCP. Instead of storing the state, Beamer uses per-connection state held by servers to perform the forwarding. Beamer was implemented on a software switch (BMv2) and on NetFPGA.

Liu et al. [168] proposed DistCache, a system that enables load balancing for storage systems through a distributed caching mechanism. DistCache enables cache allocation by leveraging independent hash functions, and provides query routing through power-of-two-choices, hence, creating a cache abstraction. The system was implemented on a hardware switch (Tofino).

Hsu et al. [169] proposed DASH, a data structure that dynamically balances data across multiple paths in the data plane. It assigns a small number of hash boundaries to each stage and maintains the hash boundaries by leveraging multiple pipeline stages and per-stage SALUs. The system was implemented on a software switch (BMv2), and results show that DASH only requires tens of packets to update weights, significantly lower than legacy approaches (WCMP).

A.3. Comparison between Switch-based and Server-based Load Balancer

Table XIV shows a comparison between switch-based and server-based load balancers. There is a significant improvement in the throughput when load balancing is offloaded to the switches; for instance, SilkRoad [165], which is a load balancing scheme in the data plane, achieves 10 billion packets per second (pps) while operating at line rate. Software load balancers on the other hand achieve a much lower throughput, nine million PPS on average. Software-based load balancers also incur additional latency overhead when processing new requests. It is relatively easy to install additional software load balancers, which makes it more scalable than switch-based load balancing schemes. Moreover, software load balancers are more flexible in assigning flow identification policies. Finally, switch-based schemes are simpler as the whole logic is expressed in a program (customized parser and match-action tables), whereas server-based balancers might require additional coordination with routers (e.g., tunneling).

TABLE XIV
SWITCH-BASED AND SERVER-BASED LOAD BALANCERS.

Feature	Switch-based	Server-based
Throughput	Higher; (e.g., SilkRoad with 6.4Tbps ASIC can achieve about 10Gpps)	Lower (e.g., 9Mpps per core [299])
Latency	Lower; sub-microseconds from ingress to egress	Higher; additional latency when processing new requests *
Scalability	Lower; connection is stored in limited SRAM	Higher
Policy flexibility	Limited; hash-based flow assignments may lead to imbalance	Flexible policies can be written in software
System complexity	Simpler; it requires a customized parser, match-action tables	More complex; it requires coordination with routers, tunneling (e.g., GRE encapsulation)

*After the first packet is processed, no additional latency is observed [299].

B. Caching

B.1. Background

Modern applications (e.g., online banking, social networks) rely on key-value stores. For example, retrieving a single web page may require thousands of storage accesses. As the number of users increases to millions or billions, the need for higher throughput and lower latency is needed. A challenge of key-value stores is the non-uniform access of items. Instead, popular items, referred to as “hot items”, receive more queries than others. Furthermore, popular items may change rapidly due to popular posts, limited-time offers, and trending events [170]. Fig. 26(a) shows a typical skew key-value store system which presents load imbalance among servers storing key-value objects. The performance of such systems may present reduced throughput and long latencies. For example, server 2

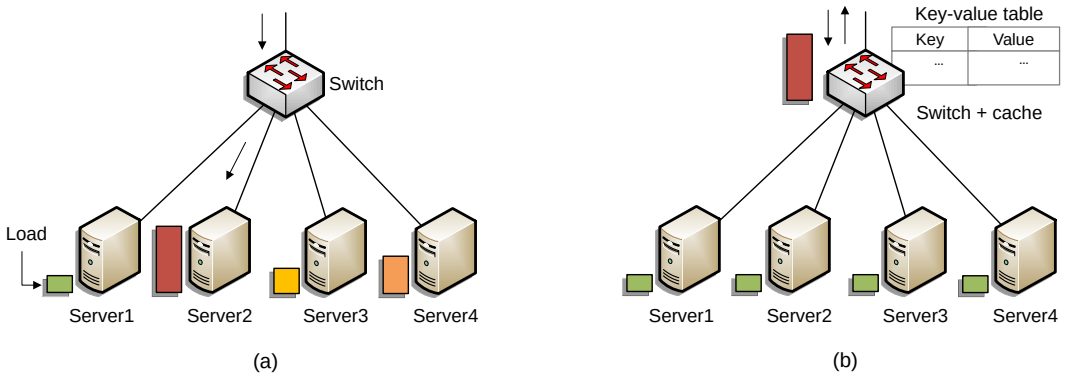


Fig. 26. Traditional software-based versus switch-based caching system.

may add substantial latency as a result of storing a hot item and being over-utilized, while server 1 is under-utilized.

B.2. Literature Review

Fig. 26(b) illustrates a system where a programmable switch receives a query before forwarding them to the server storing the key. The switch is used as "in-network cache," where the hottest items are stored. When a read request for a hot key is received, the switch consults its local table and returns the value corresponding to that key. If the key is missed (i.e., the case for non-hot keys) then the switch forwards the request to the appropriate server. When a write request is received, the switch checks its local table and evicts the entry if the key is stored there. It then forwards the request to the appropriate backend server. A controller periodically collects statistics to update the cache with the current hot items.

Jin et al. [170] proposed NetCache, an in-network architecture that uses programmable switches to store hot items and balance the load across storage nodes. NetCache efficiently detects hot key-value items, then indexes, caches and serves them in the switch data plane. Their implementation on a hardware switch (Tofino) shows that for 64,000 items that have 16-bytes keys and 128-bytes values, more than two billion queries were processed per second. Compared to commodity servers, NetCache improves the throughput by 3-10 times and reduces the latency of 40% of queries by 50%.

Cidon et al. [171] proposed AppSwitch, a packet switch that performs load balancing for key-value storage systems. Signorello et al. [300] developed a preliminary implementation of Named Data Networking (NDN) instance using P4. Grigoryan et al. [301] proposed a system that caches Forwarding Information Base (FIB) entries (the most popular entries) in fast memory in order to minimize the TCAM consumption and to avoid the TCAM overflow problem.

Liu et al. [172] proposed IncBricks, a caching fabric for key-value pairs with basic computing primitives in the data plane. The system was implemented using Cavium XPliant switches, and the results show that the latency of requests is reduced by over 30% and the throughput is tripled compared to client-side caching systems.

Zhang et al. [173] proposed B-Cache, a behavioral caching framework for programmable switches. The motivation behind B-Cache is that the performance of the data plane decreases significantly as the complexity of the P4 program and the packet processing pipeline grows. As a result, B-Cache aims at caching into a single cache match-action table, which will be used to match incoming packets. In case there is a match, the packet bypasses the original pipeline, making the performance of caching independent of the pipeline length. The system was implemented on a software switch (BMv2) and a smartNIC.

Vestin et al. [174] proposed FastReact, a system that enables caching for industrial control networks using programmable data planes. Data cached in FastReact represents sensors' data collected from Internet of Things (IoT) devices. The system also allows performing few computations such as calculating the average of sensors' values. The switch tracks the history of the last received sensors' values, and records their timestamp, which enables other devices to query recent readings. The system was implemented on a software switch (BMv2).

Woodruff et al. [175] proposed P4DNS, a Domain Name System (DNS) accelerator implemented on a programmable switch that caches DNS entries and brings computation closer to the end host. The system was implemented using NetFPGA-SUME, and results show that P4DNS improve the performance x52 compared to software-based solutions.

B.3. Comparison between Switch-based and Server-based Caching

Table XV compares the switch-based versus server-based caching schemes. The throughput when data is cached on the switch is order of magnitude larger than that of general purpose servers. The latency is also reduced by 50%, and most of it is induced by the client. The switched-based caching solves the load imbalance problem and is simpler as the whole logic is expressed in a program. Server-based caching on the other hand is more flexible regarding cache policies, as well as keys, values, and tables' sizes.

C. Telecommunication Services

C.1. Background

The evolution of the current mobile network to the emerging Fifth-Generation (5G) technology implies significant improvements of the network infrastructure. Such improvements are necessary in order to meet the Key Performance Indicators

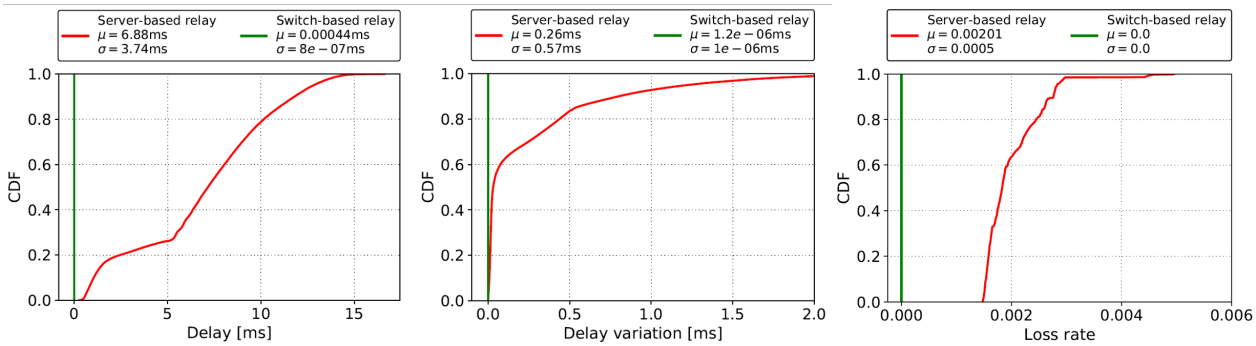


Fig. 27. CDF of delay, delay variation, and packet loss rate of 900 offloaded VoIP calls [180].

TABLE XV
SWITCH-BASED AND SERVER-BASED CACHING.

Feature	Switch-based	Server-based
Throughput	Higher; (e.g., NetCache, 2BQPS[1])	Lower; 0.2BQPS
Latency	Lower; (e.g., NetCache, 7 µs, mostly caused by the client)	Higher; 15 µs
Key size	Not flexible (limited by packet header length)	Arbitrary
Value size	Not flexible (limited by the amount of state accessed when processing a packet)	Arbitrary
Load imbalance	No	Yes
System complexity	Simpler; it requires a customized parser, match-action tables	More complex; it requires coordination with routers, tunneling (e.g., GRE encapsulation)
Table size	Limited by RAM	Arbitrary
Cache policies	Limited by table size	Arbitrary

[1]BQPS: Billion Queries Per Second.

(KPIs) and requirements of 5G [302]. 5G requires ultra-reliable low latency and jitter (microseconds-scale). As programmable switches fulfill these requirements, researchers are investigating the idea of offloading telecom-oriented VNFs running on x86 servers to programmable hardware.

C.2. Literature Review

Ricart-Sanchez et al. [176] proposed a system that uses programmable data plane to enhance the performance of the data path from the edge to the core network, also known as the backhaul, in a 5G multi-tenant network. The system was implemented using NetFPGA-SUME, and the experiments' results show that the attained QoS meets the requirements of 5G.

Ricart-Sanchez et al. [177] proposed a 5G firewall based on programmable data plane that detects, differentiates and selectively blocks 5G network traffic in the backhaul network. The aim of this system is to meet the reliability KPI of 5G which states that the network should be secured with zero downtime. The system was implemented using NetFPGA-SUME.

Palagummi et al. [178] proposed SMARTHO, a system that uses programmable switches to perform handover efficiently in a wireless network. Essentially, the switches process packets exchanged between the Central Unit (CU) and the Distributed Unit (DU) in a Next Generation RAN (NG-RAN), and perform the resource allocation for User Equipment (UE) in advance. The system was implemented on a software switch (BMv2), and results show that 18% and 25% reductions in handover time are achieved compared to legacy approaches, for two- and three-handover sequences, respectively.

Singh et al. [179] designed a P4-based element of 5G Mobile Packet Core (MPC), namely the virtual Evolved Packet Gateway (vEPG). The vEPG is responsible for merging the functions of both signaling gateway (SGW) and the Packet Data Network Gateway (PGW). The system was implemented on a hardware switch (Tofino). By offloading MPC user plane functions to programmable ASIC, the authors managed to achieve a low latency ($<2\mu$s) and jitter, scaling up to 1.7 millions active users.

Kfoury et al. [180] proposed a system for offloading conversational media traffic (e.g., Voice over IP (VoIP), Voice over LTE (VoLTE), WebRTC, media conferencing, etc.) from x86-based relay servers to programmable switches. The system emulates the behavior of the relay server which is primarily used to solve the NAT problem. The system was implemented on a hardware switch (Tofino). Results show that ultra-low latency and jitter (nanoseconds-scale) are achieved with programmable switches as opposed to x86-based relay servers where the latency and the jitter are in the milliseconds-scale. The solution also improves the packet loss rate, CPU usage of the server, Mean Opinion Score (MOS) as shown in Fig. 27, and can scale to more than one million concurrent sessions, with additional resources to spare in the switch.

Shah et al. [181] proposed TurboEPC, a system that offloads a subset of user state in mobile packet core to programmable switches in order to perform signaling in the data plane. TurboEPC offloads messages that constitute a significant portion of the total signalling traffic in the packet core, aiming at improving throughput and latency of the control plane's processing. The system was implemented on a smartNIC (Netronome Agilio).

C.3. Switch-based and server-based media relay

Offloading media traffic from general purpose servers to programmable switches greatly improves the quality of ser-

TABLE XVI
SWITCH-BASED AND SERVER-BASED MEDIA RELAYING.

Metric	Switch-based relay [180]	Server-based relay
Relay server CPU	Lower; negligible with 900 active sessions	Higher; averages at 50% for 900 active sessions
Latency	Lower; almost constant at 440ns with 900 sessions	Higher; from 0.2ms to 17ms with 900 sessions
Jitter	Lower; negligible with 900 active sessions	Higher; ranges from 100us to 3ms
Packet loss	None contributed by the switch	High; increases as the number of sessions increases
Maximum number of sessions	Higher; more than one million with additional resources to spare	Lower; thousand sessions per core before QoS degrades
Mean opinion score (MOS)	Higher; maximum MOS (4.4) with 1800 concurrent sessions	Lower; for 1800 sessions, 50% of sessions have a MOS score below 3.7
Table size	Limited by SRAM	Arbitrary
Additional functions	Limited to relaying	Arbitrary; e.g., media mix, lawful interception

vice. Table XVI shows the metrics achieved when media is relayed by a relay server versus when it is relayed by the switch, based on [180]. The results show that the latency, jitter and packet loss rates are significantly lower when media is being relayed by the switch. Not only the QoS metrics are improved, but also the maximum number of concurrent sessions. With Tofino 3.2Tbps, more than one million sessions were accommodated in the switch's SRAM, with additional resources to spare for other functionalities. On the other hand, only one thousand sessions per CPU core were handled in the server-based relay, before QoS starts to degrade. The drawback of offloading media traffic to the switch is that some functionalities are complex to be implemented in the data plane (e.g., media mixing for conference calls).

D. Publish/Subscribe

D.1. Background

Emerging network architectures (e.g., [304]) promote content-centric networking, a model where the addressing scheme is based on named data rather than named hosts. In other words, users specify the data they are interested in instead of specifying where to get the data from. A branch of content-centric networking is the publish/subscribe (pub/sub) model. The goal of the model is to provide a scalable and robust communication channel between producers and consumers of information. A large fraction of today's Internet applications follow the publish/subscribe paradigm. With the IoT, this paradigm proliferated as sensors/actuators are often deployed in dynamic environments. Other applications that use pub/sub model include instant messaging, Really Simple Syndication (RSS) feeds, presence servers, telemetry and others. Current approaches to content-centric networking use software-based middleboxes, which limits the performance in terms of throughput and latency. Recent works are leveraging programmable switches to overcome the performance limitations of software-based pub/sub middleboxes.

D.2. Literature Review

Jepsen et al. [182] presented "packet subscription", a new abstraction that generalizes the forwarding rules by evaluating stateful predicates on input packets. The authors described a compiler that generates P4 tables from logical predicates. It utilizes a novel algorithm based on Binary Decision Diagrams (BDDs) to preserve switch resources (TCAM and SRAM). The proposed abstraction support a variety of use cases such as high-level network services, caching, customized routing, pub/sub, etc. The prototype was evaluated on a hardware switch (Tofino), and the authors considered the pub/sub use case (Nasdaq's ITCH protocol). Results show that the system was able to process messages at line rate while using the full switch capacity (6.5 Tbps).

Wernecke et al. [183, 184] presented distribution strategies for content-based publish/subscribe systems using programmable switches. The authors described a system where the notification distribution tree (i.e., the subscribers that should receive the notification) is encoded in the packet headers, similar to multicast source routing. The advantage of storing the distribution tree in the packet header instead of storing it in the switch is that rules in the switches do not need to be updated when subscriptions change. The system was implemented on a software switch (BMv2), and different tree distribution strategies were evaluated.

Kundel et al. [185] implemented a publish/subscribe system on programmable switches. The system is flexible in encoding attributes/values in packet headers. The encoded data is extracted recursively by the parser in order to perform subscription matching subsequently. The system was implemented on a software switch (BMv2), and the authors assumed the results of previous work to estimate the overall forwarding delay.

D.3. Comparison between Switch-based and Server-based Pub/Sub Systems

Fig. 28 illustrates the operations of traditional software-based pub/sub systems (a) and switch-based pub/sub systems (b). Latency and its variations are significantly reduced when the switch acts as a pub/sub broker. However, the size of memory in the switch limits the amount of data to be distributed. Moreover, implementing features provided by software-based pub/sub systems such as QoS levels, session persistence, message retaining, last will and testament (notify users after a device disconnects) in hardware is challenging.

E. Summary

Table XVII summarizes the works related middlebox functions. Programmable switches offer the flexibility of customizing the data plane to enable middlebox functions. A middlebox can be defined as a device that performs functions which are beyond the standard capabilities of routers and switches. A number of works demonstrated the implementation of middlebox functions such as caching, load balancing, offloading services, and others on programmable switches.

XI. NETWORK-ACCELERATED COMPUTATIONS

Programmable switches offer the flexibility of offloading some upper-layer logic to the ASIC, referred also as in-

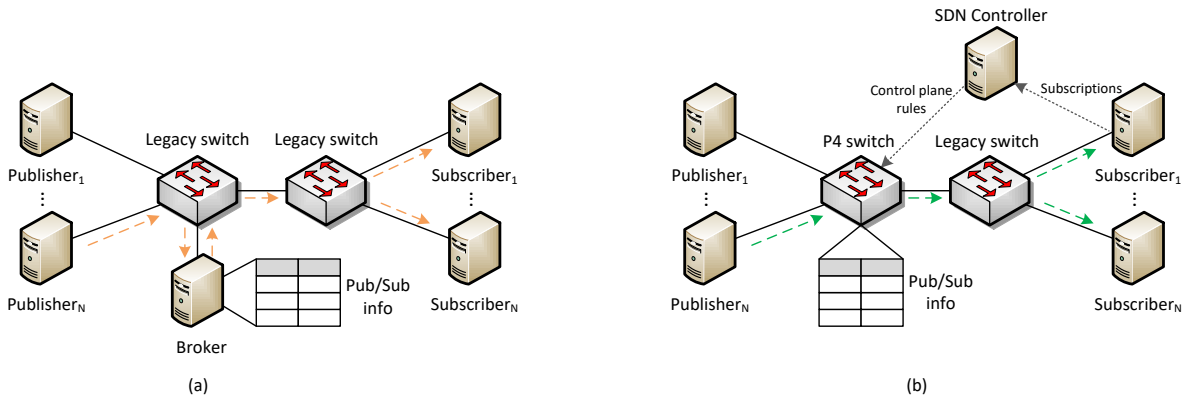


Fig. 28. (a) Traditional software-based pub/sub architecture. (b) Pub/sub implemented on a programmable switch.

network computation. Since switch ASICs are designed to process packets at terabits per second rates, in-network computation can result in an order of magnitude or more of improvement in throughput when compared to applications implemented in software. The potential performance improvement has motivated programmers to built in-network computation for different purposes, including consensus, machine learning acceleration, stream processing, and others.

The idea of delegating computations to networking devices was perceived with Active Networks [305], where packets are replaced with small programs ("capsules") that are executed in each traversed device along the path. However, traditional network devices were not capable of performing computations. With the recent advancements in programmable switches, performing computations is now a possibility.

A. Consensus

A.1. Background

Consensus algorithms are common in distributed systems where machines collectively achieve agreement on a single data value, or on the current state of a distributed system. Reliability is achieved with consensus algorithms, even in the presence of some malicious or faulty processes. Consensus algorithms are used in applications such as blockchain [306], load balancing, clock synchronization, and others [307]. Latency has always been a bottleneck with consensus algorithms as protocols require expensive coordination on every request. Lately, researchers have started investigating how programmable switches can be leveraged to operate consensus protocols in order to increase throughput and decrease latency.

TABLE XVII
SUMMARY OF MIDDLEBOX FUNCTIONS LITERATURE.

Category	Ref	Name	Description	Implementation		
				Hardware (Tofino)	Software (BMv2)	Others
Load Balancing	[164]	HULA	Stores the best path to the destination via its neighboring switch			ns-2
	[165]	SilkRoad	Provides a direct path between application traffic and servers	✓		
	[166]	MP-HULA	Best-k paths to the destination through the neighbor switches			ns-2
	[167]	Beamer	Uses per-connection state held by servers to perform the forwarding		✓	
	[168]	DistCache	Balances through a distributed caching mechanism for storage systems	✓		
	[169]	DASH	Dynamically balances data across multiple paths in the data plane		✓	
Caching	[170]	NetCache	Detects, indexes, caches, and servers hot key-value items in the data plane	✓		
	[171]	AppSwitch	Packet switch that performs load balancing for key-value storage systems			PISCES
	[172]	IncBricks	Caching fabric for key-value pair in the data plane			Cavium XPliant
	[303]	NDN.p4	Preliminary open source implementation of NDN on P4		✓	
	[301]	PFCA	Caches Forwarding Information Base (FIB) entries			N/A
	[173]	B-Cache	Improve caching performance by bypassing the original pipeline		✓	
	[174]	FastReact	Caching for industrial control networks		✓	
	[175]	P4DNS	Offloads DNS function to programmable switches			NetFPGA
Telecom	[176]	N/A	Enhances data path in the backhaul of a 5G multi-tenant network			NetFPGA
	[177]	N/A	5G network traffic firewall in the backhaul network			NetFPGA
	[178]	SMARTHO	Performs handover in Next Generation RAN (NG-RAN)		✓	
	[179]	N/A	Offloads Mobile Packet Core (MPC) user plane functions	✓		
	[180]	N/A	Offloads conversational media traffic (VoIP, VoLTE, WebRTC, etc.)	✓		
	[181]	TurboEPC	Offloads a subset of user state to perform signalling in the data plane			SmartNIC
Pub/Sub	[182]	Packet Subscription	Generalizes forwarding rules by evaluating stateful predicates	✓		
	[183] [184]	N/A	Distribution strategies for content-based publish/subscribe systems		✓	
	[185]	N/A	Publish/subscribe system on programmable switches		✓	

Fig. 29 shows a consensus model in the data plane.

A.2. Literature Review

Li et al. [186] proposed Network-Ordered Paxos (NOPaxos), a P4-based Paxos [308] system that applies replication in the data center to reduce the latency imposed from communication overhead. Unordered and completely asynchronous networks require the full implementation and complexity of Paxos. The paper suggests that the communication layer should provide a new Ordered Unreliable Multicast (OUM) primitive; that is, there is a guarantee that receivers will process the multicast messages in the same order, though messages can be lost. NOPaxos relies on the network to deliver ordered messages in order to avoid entirely the coordination. Dropped packets on the other hand are handled through coordination with the application. The solution was implemented on a hardware switch (Cavium) and the results show that the system outperforms Paxos by 54% in latency and 4.7 times in throughput.

Dang et al. [187] presented an implementation of Paxos using P4 on the data plane. The system was implemented on a software switch (BMv2), aiming to show new practical concerns and design decisions for the algorithm that were not been addressed at the time. The implementation is simplistic as the authors did not consider the resource limitations of a hardware switch. Moreover, it only implemented phase 2 logic of Paxos leaders and acceptors.

Li et al. [188] proposed Eris, a P4-based solution that avoids replication and transaction coordination overhead. It processes a large class of distributed transactions in a single round trip, without any additional coordination between shards and replicas. The main contribution of Eris compared to NOPaxos is that it establishes a consistent ordering across messages delivered to many destination shards. Eris also allows receivers to detect dropped messages. The hardware required to evaluate Eris was not commercially available at the time and thus the authors used network processors (Cavium Octeon II CN6880 [309]).

Dang et al. [189] proposed P4xos, a P4-based solution that executes Paxos logic directly in switch ASICs, without strengthening assumptions about the network (e.g., ordered delivery, packet loss, etc.). As the consensus messages are processed in the data plane, the number of hops (network and hosts) are reduced. Hence, P4xos improves both the latency and the tail-latency. Furthermore, the throughput is improved compared to hardware servers which require additional memory management and safety features (e.g., user and kernel separation). P4xos was implemented on a hardware switch (Tofino), and results show that it reduces the latency by three times compared to traditional approaches, and it can process over 2.5 billion consensus messages per second (four orders of magnitude improvement). Jin et al. [190] proposed NetChain, a P4-based system that provides scale-free sub-RTT coordination in data centers. It is strongly-consistent, fault-tolerant, and presents an in-network key-value store. NetChain uses a variant of the Paxos protocol that divides it into two parts: steady state and reconfiguration. This variant is known as Vertical Paxos, and is relatively simple to implement in the network as the division's parts can be mapped to the control plane and the data plane. The solution was implemented on a hardware switch (Tofino), and the results show that the solution provides orders of magnitude higher throughput and lower latency than traditional server-based solutions.

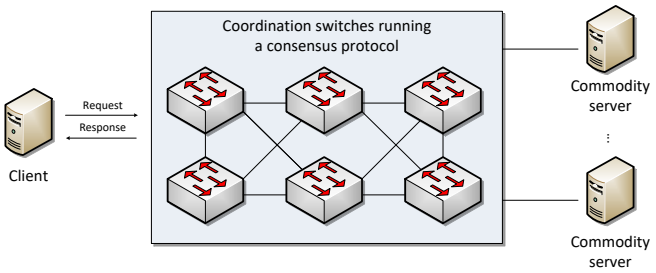


Fig. 29. Consensus protocol in the data plane model.

Dang et al. [191] proposed Partitioned Paxos, a P4-based system that separates the two aspects of Paxos, namely, agreement and execution, and optimizes them separately. The motivation behind Partitioned Paxos is that existing network-accelerated approaches do not address the problem of how replicated application can cope with the high rate of consensus messages; NOPaxos only processes 13,000 transactions per second since it presents a new bottleneck at the host side. Other systems (e.g. NetChain) are specialized replication services and can not be used by any off-the-shelf application. The proposed system was implemented on a hardware switch (Tofino), and results show that it can process over 2.5 billion consensus messages per second.

Sakic et al. [192] proposed P4 Byzantine Fault Tolerance (P4BFT), a system that is based on BFT-enabled SDN, where controllers act as replicated state machines. The system offloads the comparison of controllers' outputs required for correct BFT operations to programmable switches. Since the distance between the controllers and the processing nodes is shortened, the network load imposed by BFT operation is decreased. The solution was implemented on a software switch (BMv2) and on a smart NIC (Netronome Agilio SmartNIC). Results show that comparison of controller outputs in the hardware results in 96.4% decrease in delay compared to software-based approaches.

Han et al. [193] offloaded part of the Raft consensus algorithm [310] to programmable switches in order to improve its performance. The authors selected Raft due to the fact that it has been formally proven to be more safe than Paxos, and it has been implemented on popular SDN controllers. The system is implemented on software switches (BMv2), and only preliminary results were presented. Note that measuring latency on a software switch is not quite as representative as using a hardware switch. The authors discussed implementing the solution on a hardware switch as future work.

A.3. Network-assisted and legacy consensus comparison

Consensus algorithms have been traditionally implemented as application on general purpose CPUs. Such architecture inherently induces latency overhead (e.g., Paxos coordinator has a minimum latency of 96us [311]).

There are numerous performance benefits gained when consensus algorithms are implemented in programmable devices. When consensus messages are processed on the wire, the latency significantly decreases (Paxos coordinator had a minimum latency of 340ns [311]). Moreover, when compared to legacy consensus deployments, network-assisted consensus require fewer hops traversal.

B. Machine Learning

B.1. Background

The remarkable success of Machine Learning (ML) today has been enabled by a synergy between development in hardware and advancements in machine learning techniques. Increasingly complex ML models are being developed to handle the large size of datasets and to accelerate the training process. Hardware accelerators (e.g., GPU, TPU) were introduced to speedup the training. These accelerators are installed in large clusters and collaborate through distributed training to exploit parallelism. Nevertheless, training ML models is time consuming and can last for weeks depending on the complexity and the size of the datasets. Researchers have traditionally investigated methods to accelerate the computation process, but not the communication in distributed learning. With the advancements in programmable switches, it is now possible to accelerate the ML training process through the network.

B.2. Literature Review

Sapio et al. [194] proposed DAIET, a system that performs in-network data aggregation to minimize communication overhead of exchanging ML model updates. While the goal of this paper is to discuss what kinds of computations the network can perform, the authors did not design a complete system, nor did they address the major challenges of supporting ML applications. Moreover, their proof-of-concept presented a simple MapReduce application on a software switch (BMv2), and it is not certain whether the system can be implemented on a hardware switch.

Siracusano et al. [195] proposed N2Net, a system that runs simplified neural networks (NN) on programmable switches. While full-fledged neural networks require complex computations (e.g., multiplications and activation functions), more simple variation such as the Binary Neural Network (BNN) only requires bitwise logic functions (e.g., XNOR, POPCNT, SIGN). N2Net provides a compiler that translates a given neural network model to switching chip's configuration (P4 program). The authors did not mention on which platform N2Net was evaluated; however, based on their evaluations, they concluded that a BNN can be implemented on most current switching chips, and with small additions to the chip design, more complex models can be implemented.

Sanvito et al. [196] proposed BaNaNa Split, a solution that evaluates the conditions under which programmable switches can act as CPUs' co-processors for the processing of Neural Networks (CNN). The authors initially performed an analysis on the required inference computations in CNN and profiled the execution on general purpose CPUs. Then, by leveraging the NN's layered structure, they analyzed options for partitioning the processing of NN to offload a subset of layers from the CPU to a different processor. BaNana Split performs NN models quantization and executes said models on both CPUs and programmable switches. The solution is far from complete, and the authors evaluated a single binary fully connected layer with 4096 neurons using a network processor-based SmartNIC.

Yang et al. [313] proposed SwitchAgg, an in-network aggregation system that performs similar functions as DAIET. An improvement of SwitchAgg over DAIET is that the network architecture need not be modified. Another advantage is that it provides both high processing ability and a significant data reduction rate. Moreover, the system was implemented on a hardware platform (NetFPGA-SUME), and the results show that the job completion time can be reduced as much as 50%.

Sapio et al. [197] proposed SwitchML, a system that performs in-network aggregation to accelerate the training process of machine learning models. The most commonly used training technique for deep neural networks is synchronous stochastic gradient descent [314]. In this technique, each worker has a copy of the model that is being trained. The training is an iterative process where each iteration consists of: 1) reading the sample of the dataset and locally perform some computation-intensive learning using the worker's accelerators. This yields to a gradient vector; and 2) updating the model by computing the mean of all gradient vectors. The main motivation of this idea is that the aggregation is computationally cheap (takes 100ms), but is communication-intensive (transfer hundreds of megabytes each iteration). SwitchML uses computation on the switch to aggregate model update in the network as the workers are sending them (see Fig. 30 and Fig. 31). An advantage is that there is minimal communication; each worker sends its update vector and receives back the aggregated updates. The design challenges of this system include: 1) the limitation of storage available on the switch, addressed by using a streaming approach; 2) switches cannot perform much computations per packet, addressed by partitioning the work between the switch and the workers; 3) ML systems use floating point numbers, addressed by quantization approaches; and 4) failure recovery is needed to ensure correctness. The system is implemented on a hardware switch (Tofino), and results show that the system speeds up training by up to 300% compared to existing distributed learning approaches.

Xiong et al. [198] proposed IIsy, a system that enables programmable switches to perform in-network classification. The motivation behind this work is that ML accelerators operate at the scale of tens of microseconds to milliseconds per inference. This latency can be optimized if the inference is performed in the switches (hundreds of nanoseconds latency per packet). IIsy maps previously trained ML classification models to match-action pipelines. The authors acknowledged that this work is limited in scope as it does not address popular ML algorithms such as neural networks. Furthermore, it is bounded to the type of features it can extract (i.e., packet headers), and has accuracy limitations. IIsy tries to find a balance between the limited resources on the switch and the classification accuracy. The most common use cases for this

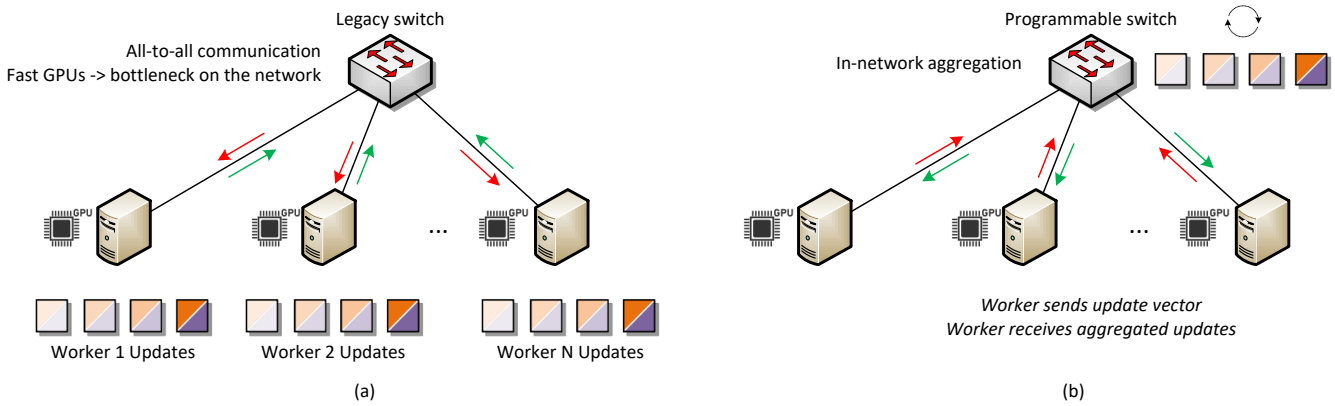


Fig. 30. (a) ML model updates in legacy networks. The aggregation process is communication-intensive and follows an all-to-all communication pattern. This means that the workers should receive all the other workers' updates. Since accelerators on end-hosts are becoming faster, the network should speed up so that it does not become the bottleneck. Therefore, it is expensive to deploy additional accelerators since it requires re-architecting the network. The red arrow in (a) shows that the bottleneck source is the network. (b) ML model updates accelerated by the network. Aggregation is performed in the network by the programmable switches while the workers are sending them. The workers do not need to obtain the updates of all other workers, hence there is minimal communication. They only obtain the aggregated model from the switch. The red arrow in (b) shows that the bottleneck source is the worker rather than the network [197, 312]

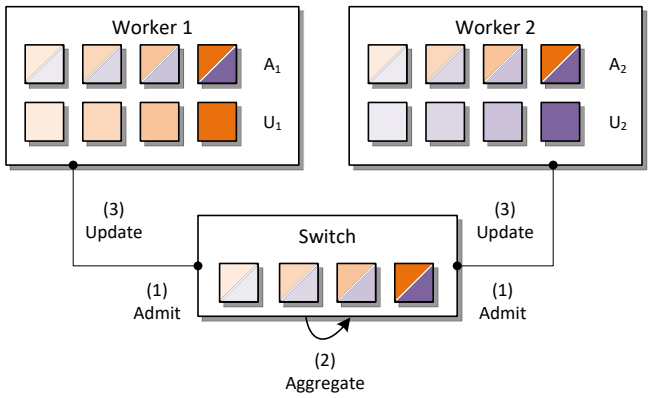


Fig. 31. In-network aggregation (source: SwitchML [197]).

system are network traffic classification and Distributed Denial of Service (DDoS) mitigation. The system was implemented on hardware (NetFPGA SUME) with four classification algorithms: decision trees, K-means, SVM, and Naïve Bayes.

Other works that use in-network computation include stream processing [199, 200], parallel processing [201], string searching [202], erasure coding [203], and in-network lock managers [204].

C. Comparison between Switch-based and Server-based ML

Table XVIII shows a comparison between switch-based and server-based ML approaches. ML works that were extracted from the literature can be divided into two main categories: 1) expedited inference in the data plane, and 2) accelerated training in the network. The main advantage of switch-based over server-based inference is the ability to execute at line rate, and hence provides faster results to the clients. Performing complex computations in the switch is achieved through estimations, and hence is limited. Moreover, the SRAM capacity of the switch is small, impeding the storage of large models. Such limitations are not problematic with server-based inference approaches.

Distributed training can be significantly faster when aggregations are offloaded to a centralized switch. However, due the small capacity of the switch memory, it is not possible to store the whole model update at once. Additionally, encrypted traffic remains a challenge when inference or training is handled by the switch.

D. Summary

Table XIX summarizes the works pertaining to in-network accelerated computations. Accelerating computations by way

TABLE XVIII
SWITCH-BASED AND SERVER-BASED ML APPROACHES.

Feature	Inference		Training	
	Switch-based	Server-based	Switch-based	Server-based
Speed	Faster, inference at line rate	Slower	Faster, aggregations at line rate	Slower; aggregations on an x86 server
Complex computations support	Lower, basic arithmetic and bitwise logic function	Higher	Lower	Higher
Communication overhead	Low	Low	Lower, switch is the centralized aggregator	Higher, updates are exchanged with a remote aggregator
Storage	Lower	Higher	Lower, update is not stored entirely at once	Higher
Encrypted traffic	Difficult	Easy	Difficult	Easy

TABLE XIX
SUMMARY OF ACCELERATED COMPUTATIONS LITERATURE.

Category	Ref	Name	Description	Implementation		
				Hardware (Tofino)	Software (BMv2)	Others
Consensus	[186]	NOPaxos	The network delivers ordered messages in Paxos			Cavium
	[187]	Paxos made switch-y	Simplistic implementation of Paxos on P4		✓	
	[188]	Eris	Avoids replication and transaction coordination overhead			Cavium
	[189]	P4xos	Paxos logic in switch ASICs without assumptions on the network	✓		
	[190]	NetChain	Scale-free sub-RTT coordination in data centers	✓		
	[191]	Partitioned Paxos	Optimizes Paxos' agreement and execution aspects separately	✓		
	[192]	P4BFT	Byzantine Fault Tolerance comparison operations on the switch		✓	SmartNIC
	[193]	N/A	Offloads part of the Raft consensus algorithm to the switch		✓	
Machine Learning	[194]	DAIET	Minimizes communication overhead of exchanging ML model updates		✓	
	[195]	N2Net	Simplified neural networks (binary) on programmable switches			N/A
	[196]	BaNaNa Split	Switches work as CPUs co-processors for the processing of NNs			SmartNIC
	[313]	SwitchAgg	Aggregation to minimize overhead without modifying the network			NetFPGA
	[197]	SwitchML	Aggregates model updates in-network as the workers are sending them	✓		
	[198]	IIsy	Maps trained ML classification models to match-action pipelines			NetFPGA

of leveraging programmable switches is becoming a trend in data centers and backbone networks. Although switches only support basic and limited operations, it was shown in the literature that the performance of various tasks (e.g., consensus, training models in machine learning), could significantly improve if computations are delegated to the network.

XII. INTERNET OF THINGS (IOT)

The Internet of Things (IoT) is a novel paradigm in which pervasive devices equipped with sensors and actuators collect physical environment information and control the outside world. IoT applications include smart water utilities, smart grid, smart manufacturing, smart gas, smart metering, and many others. Typical IoT scenarios entail a large number of devices periodically transmitting their sensors' readings to remote servers. Data received on those collectors is then processed and analyzed to assist organizations in taking data-driven intelligence decisions.

A. Aggregation

A.1. Background

Since IoT devices are constrained in size and processing capabilities, they typically generate packets that carry small payloads (e.g., temperature sensor readings). While such packets are small in size, their headers occupy a significant portion of the total packet size. For instance, Sigfox Low-Power Wide Area Network (LPWAN) [315] can support a maximum of 12-bytes payload size per packet. The overhead of headers is 42-bytes (Ethernet 14-bytes + IP 20-bytes + UDP 8-bytes), which represent approximately 78% of the packet total size. When numerous devices continuously transmit IoT packets, a significant percentage of network bandwidth is wasted on transmitting these headers.

Packet aggregation is a mechanism in which the payloads of small packets are aggregated into a single larger packet in order to mitigate the bandwidth overhead caused by transmitting multiple headers. Legacy packet aggregation mechanisms operate on the CPUs of servers or on the control plane of switches [316–321]. While legacy mechanisms reduce the overhead of packet headers, they unquestionably increase the end-to-end latency and decrease the throughput. As a result, some studies have suggested aggregating only packets that are not real-time.

A.2. Literature Review

Wang et al. [205, 206] presented an approach where small IoT packets are aggregated into a larger packet in the switch data plane. Upon receiving a packet, the P4 switch parses its headers and identifies whether the packet is an IoT packet. If the packet was identified as an IoT packet, the switch parses and extracts the payload. Afterwards, the payload is stored in switch registers along with some other metadata, and the packet is dropped. The goal of performing this aggregation is to minimize the bandwidth overhead of packets' headers. Once packets are aggregated, the resulting packet is sent across the WAN to reach the remote server. Before the packet reaches the server, it is disaggregated by another P4 switch situated close to the server and several packets identical to the original ones are generated. An important observation is that the aggregation/disaggregation processes are transparent to both the IoT devices and the servers; hence, no modifications are required on either end. The authors implemented the system on both a software switch (BMv2) and a hardware switch (Tofino). Results show that the aggregation process runs at line rate (100Gbps per port). However, the disaggregation has a lower throughput, 30% of the port capacity, due to the expansion of a number of bytes after the packet disaggregation process is performed. Before implementing the solution on a hardware switch to test the throughput of packet aggregation/disaggregation, the authors analyzed mathematically the properties of the generated stream of aggregated packets [207]. The authors proposed to provide a small buffer at the input ports in order to reduce the drop probability in the disaggregation process.

Madureira et al. [208] proposed IoTP, a layer-2 commu-

TABLE XX
SWITCH-BASED AND SERVER-BASED PACKET AGGREGATION.

Feature	Switch-based (ASIC)	Server-based (CPU)
Throughput	Higher; (e.g., [205], 100Gbps, i.e., max capacity)	Lower; (e.g., [205], 2.58Gbps)
Latency and Jitter	Lower;	Higher;
Count of packets to be aggregated	Not flexible (limited by the switch SRAM)	Arbitrary
Amount of data to be aggregated	Not flexible (limited by the switch SRAM, parsing capacity)	Arbitrary

nication protocol that enables the aggregation of IoT data in programmable switches. The solution gathers network information that includes the Maximum Transmission Unit (MTU), link bandwidths, underlying protocol, and delays. These properties are used to empower the aggregation algorithm. The authors also considered tuning the solution so that IoT data retrieval can be prioritized. The system was implemented on a software switch (BMv2), and the experiments consisted of varying number of IoT devices with varying transmissions' frequencies, different amounts of data to be aggregated, and other network characteristics. Results indicate that the proposed system improves network efficiency and is able to control data aggregation-induced delays.

A.3. Comparison between Server-based and Switch-based Aggregation

Table XX shows a comparison between switch-based and server-based packet aggregation. When aggregation is performed on the switch (ASIC), the throughput is higher while the latency and jitter are lower than that of the server-based approaches (e.g., switch CPU or x86-based server). On the other hand, the server-based aggregation has more flexibility in defining the number of packets and the amount of data that can be aggregated.

B. Service Automation

B.1. Background

Low-power low-range IoT communication technologies (e.g., Bluetooth Low Energy (BLE) [322], Zigbee [323], Z-wave [324]) typically follow a peer-to-peer model. IoT devices in such technologies can be divided into two distinct types, *peripheral* and *central*. Peripheral devices, which consist of sensors and actuators, receive commands and execute subsequent actions. Central devices on the other hand run applications that analyze information collected from peripheral devices and subsequently issue commands.

The interconnection of devices and services can follow a Peer-to-Peer (P2P) model or a cloud-centric approach. In the P2P model, the automation service runs on the central device which processes and analyzes sensor data published by peripheral devices in order to issue commands. The main advantages of the P2P include the low end-to-end latency and the subtle power consumption as devices are physically close to each other. The drawbacks of the P2P model include poor scalability, short reachability, and inflexibility of policy enforcement.

TABLE XXI
SWITCH-BASED, P2P, AND CLOUD SERVICE AUTOMATION.

Feature	Switch-based	Peer-to-peer	Cloud-based
Latency	Low	Low	High
IoT energy	Low	Low	High
Scalability	High	Low	High
Reachability	High	Low	High

The cloud-centric model addresses the limitations of the P2P model by adding a gateway node that connects peripheral devices to a middleware hosted on the cloud (Internet). While this approach solves the poor scalability and the policy enforcement flexibility issues, it incurs additional delays and jitters in collecting and reacting to data. Moreover, the middleware represents a single point of failure which can shutdown the whole service in the event of an outage.

With programmable switches, researchers are investigating in-network approaches to manage transactional relationships between low-power, low-range IoT devices.

B.2. Literature Review

Uddin et al. [209] proposed Bluetooth Low Energy Service Switch (BLESS), a programmable switch that automates IoT applications services by encoding their transactions in the data plane. It maintains link-layer connections to the devices to support P2P connectivity. The data plane operations are performed at the Attribute Protocol (ATT) service layer which consists of three operations: read attributes, write attributes, and attributes' notification. BLESS parses ATT packets, then processes and forwards them to the devices. The control plane on the other hand is responsible for address assignment, device and service discovery, policy enforcement, and subscription management. The switch was implemented on a software switch (PISCES), and the results show that BLESS combines the advantages of P2P and the cloud-center approaches. Specifically, it achieves small communication latency, low device power consumption, high scalability, and flexible policy enforcement.

The same authors proposed Muppet [210], an extension to BLESS to support multiple non-IP protocols. The system studied two popular IoT protocols, namely BLE and Zigbee. Being in the middle, Muppet switch is responsible for translating actions (e.g., on/off switch of a light bulb) between Zigbee and BLE protocols, as well as logging important events to a database which resides on the Internet via the Hypertext Transfer Protocol (HTTP). Another difference from BLESS is that the implementation of Muppet's control plane leverages ONOS controller with Protocol Independent (PI) framework.

B.3. Comparison between Server-based and Switch-based Service Automation

Table XXI shows a comparison between switch-based, P2P, and cloud-based service automation. Generally, the switch-based approach overcomes the limitations of both approaches. It achieves the low energy and latency characteristics of P2P

TABLE XXII
SUMMARY OF INTERNET OF THINGS (IOT) LITERATURE.

Category	Ref	Name	Description	Implementation		
				Hardware (Tofino)	Software (BMv2)	Software (PISCES)
Aggregation	[205, 206]	N/A	Small IoT packets are aggregated into a larger packet	✓		
	[207]	N/A	Mathematical analysis of the stream of aggregated packets		✓	
	[208]	IoTP	L2 communication protocol that aggregates IoT data		✓	
Service Automation	[209]	BLESS	Automates services by encoding transactions in the data plane			✓
	[210]	Muppet	Extension to BLESS to support multiple non-IP protocols			✓

while increasing scalability and reachability.

C. Summary

Table XXII summarizes the works pertaining to programmable data plane in the context of IoT. There exist broadly two categories, namely, packets aggregation and service automation. The goal of packet aggregation is to minimize the overhead of IoT packets' headers. Typically, headers in IoT packets represent a significant portion of the whole packet size. By aggregating several packets into a single packet, the bandwidth overhead is reduced. Moreover, since such aggregation is performed entirely in the data plane, there is no additional latency and jitter. With respect to service automation, the goal is to automate IoT applications services by encoding their transactions in the data plane while improving scalability, reachability, energy consumption, and latency. Other works that involve IoT include flowlet-based stateful multipath forwarding [325] and SDN/NFV-based architecture for IoT networks [326].

XIII. CYBERSECURITY

Extensive research efforts have been devoted on deploying programmable switches to perform various security-related functions in the data plane. Such functions include heavy hitter detection, traffic engineering, DDoS attacks detection and mitigation, anonymity, and cryptography.

A. Heavy Hitter

A.1. Background

Heavy hitters are a small number of flows that constitute most of the network traffic over a certain amount of time. They are defined depending on the port speed, network RTT, traffic distribution, application policy, and others. Heavy hitters increase the flow completion time for delay-sensitive mice flows, and represent the major source of congestion. It is important to promptly detect heavy hitters in order to react to them; for instance, redirect them to a low priority queue, perform rate control and traffic engineering, block volumetric DDoS attacks, and diagnose congestion. Traditionally, packet sampling technique (e.g., NetFlow) was used to detect heavy hitters. The main problem with such technique is the limited accuracy due to the CPU and bandwidth overheads of processing samples in the software. Advancements in programmable switches paved the way to detect heavy hitters in the data plane, which is not only orders of magnitude faster than sampling, but also enables additional applications (e.g., flow-size aware routing).

A.2. Literature Review

Sivaraman et al. [211] proposed HashPipe, a heavy hitter detection algorithm that operates entirely in the data plane. It detects the k-th heavy hitter flows within the constraints of programmable switches while achieving high accuracy. The programmable switch stores the flows identifiers and the counts of heavy flows in a pipeline of hashtables. HashPipe adapts the space saving algorithm which is described in [327]. The system was implemented on a software switch (BMv2), and was evaluated using an ISP trace provided by CAIDA (400,000 flows). Results show that HashPipe needed only 80KB of memory to identify the 300 heaviest flows, with an accuracy of 95%.

Harrison et al. [212] proposed a network-wide distributed heavy-hitter detection scheme based on programmable switches. Unlike HashPipe which considers a single switch, this solution tracks network-wide heavy hitters. Tracking network-wide heavy hitter is important as some applications (e.g., port scanners, superspreaders, etc.) cannot go undetected within a single location. Moreover, aggregating the results of switches separately for detecting heavy hitter is not sufficient; flows might not exceed a threshold locally, but the total volume is sizable. The system was implemented on a hardware switch (Tofino), and results show that the method significantly reduces the number of messages exchanged while maintaining a 100% precision.

Kučera et al. [213] proposed Elastic Trie, a solution that detects hierarchical heavy hitters, in-network traffic changes, and superspreaders in the data plane. Elastic Trie consists of a hashtable-based prefix tree that expands or collapses to focus only on the prefixes that grabs a large share of the network. The data plane informs the control plane about high-volume traffic clusters in an event-based push approach only when some conditions are met. The system was implemented on a software switch (BMv2), and the results show that the solution detects hierarchical heavy hitters with 95% precision, while only consuming few kilobytes of memory.

Basat et al. [328] proposed PRECISION, a heavy hitter detection algorithm for high-throughput programmable network switches. PRECISION probabilistically recirculates a fraction of packets for a second pipeline traversal. The recirculation idea greatly simplifies the access pattern of memory without significantly degrading throughput. The system was implemented on the software version of Tofino (Tofino model).

Ding et al. [214] proposed an approach for incrementally deploying programmable switches in a network consisting of

legacy devices with the goal of monitoring as many distinct network flows as possible. The solution exploits the HyperLogLog algorithm [329] which approximates the number of distinct elements in a multi-set. The solution is capable of detecting network-wide heavy hitters by only using partial input from the data plane. The system was implemented on a software switch (BMv2) and emulation results show that it achieves high precision compared to state-of-the-art solutions.

A.3. Comparison between P4-based and traditional heavy hitter detection

The main advantage of heavy hitters detection schemes in the data plane over sampling-based approaches is the ability to operate at line rate. This means that every packet is considered in the detection algorithm, which improves accuracy and the speed of detection. Moreover, additional applications that exploit reactive processing can be implemented. For instance, switches can perform a flow-size aware routing method to redirect traffic upon detecting a heavy hitter.

B. Cryptography

B.1. Background

Performing cryptographic functions in the data plane is useful for a variety of applications (e.g., protecting the layer-2 with cryptographic integrity checks and encryption, mitigating hash collisions, etc.). Computations in cryptographic operations (e.g., hashing, encryption, decryption) are known to be complex and resource-intensive. The supported operations in switch targets and in the P4 language are limited to basic arithmetic (e.g., additions, subtractions, bit concatenation, etc.). Recently, a handful of works have started studying the possibility of performing cryptographic functions in the data plane.

B.2. Literature Review

Scholz et al. [232] presented prototype implementations of cryptographic hash functions in three different P4 target platforms (CPU, SmartNIC, NetFPGA SUME). P4 currently implements hash functions that do not have the characteristics of cryptographic hashing. For example, Cyclic Redundancy Check (CRC), which is commonly used in P4 targets, is originally developed for error detection. CRC can be easily implemented in embedded hardware, and is computationally much less complex than cryptographic hash functions (e.g., Secure Hash Algorithm (SHA)-256); however, it is not secure and has a high collision rate. The work in [232] argues on the need to implement cryptographic hash functions in the data plane to mitigate potential attacks targeting hash collisions. Evaluation results show that 1) implementing cryptographic hash functions on CPU is easy, but has high latency (several milliseconds); 2) SmartNICs has the highest throughput, but can only process packets up to 900 bytes; and 3) NetFPGA has the lowest latency, but cannot be integrated using native P4 features. The authors found that the performance of hashing is highly dependent on the application, the input type, and the hashing algorithm, and therefore there is no single solution that fits all requirements. However, P4 targets should benefit from the characteristics of each solution (CPU, SmartNICs, FPGA, and ASICs) to implement cryptographic hashing.

Hauser et al. [233] proposed P4-IPsec, an attempt to implement host-to-site IPsec in P4 switches, which act as IPsec gateways. For simplification, only Encapsulating Security Payload (ESP) in tunnel mode with different cipher suites is implemented. The Security Policy Database (SPD) and the Security Association Database (SAD) are represented as match-action tables in the P4 switch. To avoid complex key exchange protocols such as the Internet Key Exchange (IKE), this work delegates runtime management operations to the control plane. Moreover, since encryption and decryption are not supported by P4, the authors relied on user-defined P4 externs to perform complex computations. Note that implementing user-defined externs is not applicable for ASIC (e.g., Tofino), and consequently, the main CPU module of the switch is used for performing encryption/decryption computations, at the cost of increased latency and degraded throughput. The solution was implemented in three platforms: software switch (BMv2), FPGA (NetFPGA SUME), and ASIC (Tofino).

Another work by the same authors [234] proposed P4-MACsec, an implementation of MACsec on P4 switches. MACsec is an IEEE standard for securing Layer 2 infrastructure by encrypting, decrypting, and performing integrity checks on packets. In this work, a controller continuously monitor the topology in order to setup MACsec on the switches' links. As a use case, this work proposes a novel mechanism to encrypt LLDP packets for link discovery and automated deployment of MACsec link protection. The authors implemented the MACsec functions as P4 externs within a P4 software switch target. The goal of this work is to provide insights on why P4 switches should implement security as native functions.

Chen et al. [235] implemented the Advanced Encryption Standard (AES) protocol on programmable switches using scrambled lookup tables. AES is one of the most widely used symmetric cryptography algorithms that applies several encryption rounds on 128-bit input data blocks. The idea of the proposed system is to apply permuted lookup tables by using an encryption key. The authors found that a single switch pipeline is capable of performing two AES rounds. Consequently, the system leverages *packet recirculation* technique which reinjects the packet into the pipeline. By doing so, it is possible to complete the 10 rounds of encryption required by the AES-128 algorithm by using five pipeline passes. Note that recirculation uses loopback ports and hence is limited by their bandwidth. The implementation on Tofino chip shows that $\approx$ 10Gbps throughput. The authors argued that this throughput is sufficient to support various in-network security applications. Nevertheless, it is possible to enhance the throughput by configuring additional physical ports as loopback ports.

B.3. Comparison between in-network and contemporary cryptography

Cryptographic primitives often require performing complex arithmetic operations on data. Implementing such computations on general purpose servers is simple; memory and processing units are not constrained. The literature has shown

that there is a need to implement cryptographic functions in the data plane. For instance, cryptographic hash functions can significantly improve existing data plane applications with respect to collisions; encryption can protect confidential information from being exposed to the public. However, switches have limitations when it comes to computing. Supported hash functions in P4 are non-cryptographic (e.g., CRC), and therefore, produce collisions when the table is not large. Consequently, researchers are continuously investigating techniques to perform such operations in the data plane.

C. Privacy and Anonymity

C.1. Background

Packets in a network carry information that can potentially identify users and their online behavior. Therefore, user privacy and anonymity have been extensively studied in the past (e.g., ToR and onion routing [330]). However, existing solutions have several limitations: 1) poor performance since overlay proxy servers are maintained by volunteers and have no performance guarantees; 2) deployability challenges; some solutions require modifying the whole Internet architecture, which is highly unlikely; 3) no clear partial deployment pathway; and 4) most solutions are software-based. Consequently, recent works started investigating methods that exploit programmable switches to develop partially-deployable, low-latency, and light-weight anonymity systems.

With regards to anonymity and privacy in the network, new class of attacks which target the topology, requires the attacker to know the topology and understand it's forwarding behavior. Such attacks can be mitigated by obfuscating (hiding) the topology from external users. P4-based schemes are also being developed to achieve this goal.

C.2. Literature Review

Meier et al. [229] proposed NetHide, a P4-based solution that obfuscates network topologies to mitigate against topology-centric attacks such as Link-Flooding Attacks (LFAs). The solution is scalable and preserves the usefulness of path tracing tools (e.g., traceroute). The solution formulates network obfuscation as a multi-objective optimization problem, and uses accuracy (hard constraints) and utility (soft constraints) as metrics. The system then uses ILP solver and heuristics. The P4 switches in this system capture and modify tracing traffic at line rate. The specifics of the implementation was not disclosed, but the authors claim that the system was evaluated on realistic topologies (more than 150 nodes), and more than 90% of link failures were detected by operators, despite obfuscation.

Moghaddam et al. [228] proposed Practical Anonymity at the NEtwork Level (PANEL), a lightweight and low overhead in-network solution that provides anonymity into the Internet forwarding infrastructure. The solution overcomes the performance limitations of popular anonymity systems (e.g., Tor), and does not require modifying entirely the Internet routing and forwarding protocols as proposed in [331] and [332]. Partial deployment is possible as PANEL can coexist with legacy devices. The solution involves: 1) source address rewriting to hide the origin of the packet; 2) source information normalization (IP identification and TCP sequence randomization) to mitigate against fingerprinting attacks; and 3) path information hiding (TTL randomization) to hide the distance to the original sender at any given vantage point. The solution was implemented on a hardware switch (Tofino) and results show that PANEL achieves 96% of the switch's throughput and only adds a low-latency overhead.

Kim et al. [230] proposed Online Network Traffic Anonymization System (ONTAS), a system that anonymizes traffic online using P4 switches. The system overcomes the limitations of existing systems which either requires network operators to anonymize packet traces before sharing them with other researchers and analysts, or anonymize traffic online but with significant overhead. ONTAS provides a policy language used by operators for expressing anonymization tasks, which makes the system flexible and scalable. The system was implemented and tested on a hardware switch (Tofino), and results show that ONTAS entails 0% packet processing overhead and requires half storage compared to existing offline tools. A limitation of this system is that it does not anonymize TCP/UDP field values. Another limitation is that it does support concurrently applying multiple privacy policies.

Datta et al. [231] proposed Surveillance Protection in the Network Elements (SPINE), a system that anonymizes traffic by concealing IP addresses and relevant TCP fields (e.g., sequence number) from adversarial Autonomous Systems (ASes) on the data plane. SPINE does not require cooperation between switches and end-hosts, but assumes that at least two entities (typically two ASes or two ISPs) are trusted. Fig. 32 shows the SPINE architecture. The solution encrypts the IP addresses before the packets enter the intermediary ASes. Therefore, adversarial devices only see the encrypted addresses in the headers. It also encrypts the TCP sequence and ACK numbers to mitigate against attributing packets to flows. SPINE transforms IPv4 headers into IPv6 headers when packets leave the trusted entity and restore the IPv4 headers upon entering the trusted entity. These operations enable routing to be performed in intermediary networks. The encrypted IPv4 address is inserted in the last 32-bits of the IPv6 destination address. The encryption works by XORing the IP address with the hash of a pre-shared key and a nonce. The system uses SipHash since it is easily implemented in the data plane. The system was implemented on a software switch (BMv2) and the authors tested its operation and correctness in a Mininet-emulated environment.

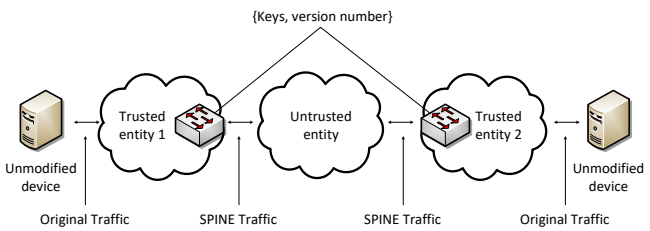


Fig. 32. SPINE architecture [231].

C.3. Privacy and Anonymity in Switch-based and Legacy Systems

Contemporary approaches that provide privacy and anonymity in the Internet uses special routing overlay networks to hide the physical location of each node from other participants (e.g., Tor). Such approaches have performance limitations as proxy servers (overlays) are maintained by volunteers and have no performance guarantees. Moreover, they often require performing advanced encryption routines to obfuscate from where the packet is originated (e.g., onion routing technique used by Tor involves encapsulating messages in several layers of encryption) . On the other hand, approaches that are based on programmable switches often rely on headers modification and simplified encryption and hashing to conceal information (e.g., SPINE [231]).

D. Access Control

D.1. Background

The selective restriction to access digital resources is known as access control in cybersecurity. Typically, access control begins with "authentication" in order to verify the identity of a party. Afterwards, "authorization" is enforced through policies to specify access rights to resources. To authenticate parties, methods such as passwords, biometric analysis, cryptographic keys, and others are used. With respect to authorization, methods such as ACL are used to describe what operations are allowed on given objects.

With the advent of programmable switches, it is now possible to delegate authentication and authorization to the data plane. As a result, access can be promptly granted or denied at line rate, before reaching the target server. A clear advantage of this approach is that servers are no longer busy processing access verification routines, which increases their service throughput.

D.2. Literature Review

Datta et al. [215] presented P4Guard, a P4-based configurable firewall that acts based on predefined policies set by the controller. The system is implemented on a software switch (BMv2).

Almain et al. [216] proposed delegating the authentication of end hosts to the data plane. The method is based on port knocking, in which hosts deliver a sequence of packets addressed to an ordered list of closed ports. If the ports match the ones configured by the network administrators, then end host is authenticated, and subsequent packets are allowed. The system was implemented on a software switch (BMv2).

Kang et al. [217] presented a scheme based on P4 switches to implement context-aware security policies. The policies are applicable to enterprise and campus networks with diverse devices, i.e., Bring Your Own Device (BYOD) (e.g., laptops, mobile devices, tablets, etc.). The scheme dynamically enforces access control to users based on contexts (e.g., if the user's device uses Secure Shell (SSH) 2.0 or higher, then the switch forwards the packets of this flow. Otherwise, the switch drops the packets). The scheme requires user devices to run an application which communicates with the switch using a custom protocol (context packets). The context packets are generated on a per-flow basis. The switch tracks flows using a match action table and registers at the data plane. Actions over a packet are dropping, allowing, and forwarding to other appliances for deep packet inspection. Data packets are not modified. Evaluations show that the proposed approach can operate (install new flows in the and update rules) with a minimum latency, even under heavy DoS attacks. On the other hand, such attacks can decimate similar SDN-based systems. One of the main drawbacks of the proposed system is the lack of authentication, integrity, and confidentiality of the context packets. Thus, the system can be subject to attacks such as snooping (i.e., eavesdropping) on communication between user devices and the switch, impersonation, and others. The system was implemented on a hardware switch (Tofino).

Bai et al. [218] presented P40f, a tool that performs OS fingerprinting on programmable switches, and consequently, applies security policies at line rate. OS fingerprinting is useful for detecting OS vulnerabilities and administrating OS-related policies to block, limit the rate, or redirect traffic. Fingerprinting can be performed actively by sending probe packets, or passively by monitoring existing network traffic. Passive fingerprinting tools have various advantages over active fingerprinting. As no probe packets are generated in passive approaches, there is no additional network load. Moreover, it is possible to fingerprint hosts outside the network, as well as BYOD devices. The main motivation behind this work is that software-based passive fingerprinting tools (e.g., p0f [333]) are not practical nor sufficient with large amounts of traffic on high-speed links. Furthermore, out-of-band monitoring systems cannot promptly take actions (e.g., drop, forward, rate-limit) on traffic at line rate. P40f was implemented on a software switch (BMv2) and validated against p0f with simulated packet traces.

D.3. Comparison between switch-based and server-based access control

Controlling access to resources often starts with authentication. While server-based approaches are more flexible in the methods of authentication they can provide, they typically require client connections to reach the server before the communication starts. In switch-based approaches, the authentication can be done in-network at the edge, eliminating unnecessary latency incurred from traversing the network and from software processing.

Access to resources can be controlled after fingerprinting end-hosts OSs. Software-based passive fingerprinting tools cannot keep up with the high load (gigabits/s links). The literature has shown that tools lead to 38% degradation in throughput [334]. Additionally, such tools are out-of-band, meaning that it is not possible to apply policies on traffic (e.g., after fingerprinting an OS). On the other hand, switch hardware is able to perform OS fingerprinting and apply security policies at line rate.

Context-aware policies applied on nodes (clients/servers) have local visibility. A newer approach is to use a centralized SDN controller (e.g., [335]), but such scheme is vulnerable to control plane saturation attacks and is subject for delay

increases. Switch-based schemes on the other hand are able to provide access control at line rate.

E. Attacks Detection and Mitigation

E.1. Background

DDoS attacks remain among the top security concerns despite the continuous efforts towards the development of their detection and mitigation schemes. This concern is exacerbated not only by the frequency of said attacks, but also by their high volumes and rates. Recent attacks (e.g. [336, 337]) reached the order of terabits per seconds, a rate that existing defense mechanisms cannot keep with.

There are two main concerns with existing defense methods handled by end-hosts or deployed as middlebox functions on x86-based servers. First, they dramatically degrade the throughput and increase latency and jitter, impacting the performance of the network. Second, they present severe consequences on the network operation when they are installed at the last mile (i.e., far from the edge).

The escalation of volumetric DDoS attacks and the lack of robust and efficient defense mechanisms motivated the idea of architecting defenses into the network. Up until recently, in-network security methods were restricted to simple access control lists encoded into the switching and routing devices. The main reason is that the data plane was fixed in function, impeding the capabilities of developing customized and dynamic algorithms that can assist in detecting attacks. With the advent of programmable data planes, it is possible to develop systems that detect and mitigate volumetric attacks without imposing significant overhead.

E.2. Literature Review

Li et al. [219] presented NETHCF, a P4-based Hop-Count Filtering (HCF) defense mechanism against spoofed IP traffic. HCF schemes filter spoofed traffic with an IP-to-hop-count mapping table. Traditional HCF-based schemes are implemented on end-hosts, which delays the filtering of spoofed packets and increases the bandwidth overhead. Moreover, since traditional schemes are implemented in server-based middleboxes, low latency and minimal jitter are hard to achieve. NETHCF aims at filtering spoofed traffic in-network. It decouples the HCF defense into a *cache* running in the data plane and a *mirror* in the control plane. The cache serves the legitimate packets at line rate, while the mirror processes the missed packets, maintains the IP-to-hop-count mapping table, and adjust the state of the system based on network dynamics. The system was implemented on a hardware switch (Tofino), and results show that NETHCF can filter spoofed traffic in an adaptive manner at line rate, with minimal overhead.

Xing et al. [220] proposed FastFlex, an abstraction that architects defenses into the network paths. FastFlex enables and disables defenses based on changing attacks, hence, it self-adapts to the observed network behavior. FastFlex tackles the following key challenges that are faced when programming defenses in the data plane: 1) resource multiplexing; 2) optimal placement; 3) distributed control; and 4) dynamic scaling. As a case study, the authors studied link-flooding attacks and considered packet-dropping defense, routing around congestion, and topology obfuscation. The system was implemented on a software switch (BMv2).

Kang et al. [221] presented an automated approach for discovering sensitivity attacks targeting the data plane programs. Sensitivity attacks in this context are intelligently crafted traffic patterns that exploit the behavior of the P4 program. For instance, a load balancer that balances traffic by hashing packet headers without cyrptographic support (e.g., modulo operator on the number of available paths) can be tricked by an attacker that craft skewed traffic patterns. This results in traffic being forwarded to a single path, leading to congestion and DoS. The proposed system in [221] accepts as input the P4 program (e.g., load balancing program), performs probabilistic symbolic execution to quantify behavior probabilities, generates traces that trigger anomalous behaviors, and synthesises monitors to detect malicious patterns. The system was evaluated on a software switch (BMv2), and results show that the system was able to discover malicious traffic patterns for a simple data plane system.

Febro et al. [222] proposed a distributed SIP DDoS defense mechanism based on programmable switches. The data plane maintains counters for the SIP INVITEs (initiation of a sessions) and SIP REGISTER (registration of an end user to the SIP server). A controller pulls these counters on a fixed time interval, and evaluates whether an attack took place by comparing to a threshold. Upon attack detection, the controller instructs the data plane to drop subsequent packets. The system was implemented on a software switch (BMv2).

Lapolli et al. [223] implemented a mechanism to perform real-time DDoS attack detection in the data plane. The proposed system estimates the entropies of source and destination IP addresses of incoming packets for consecutive partitions (observation windows). Such entropies will be used to compute anomaly detection thresholds. Since P4 does not support binary logarithm function, the authors assumed a fixed value for the observation window size. The system was implemented on a software switch (BMv2), and results show that design has low memory footprint.

Mi et al. [224] proposed ML-Pushback, a system based on Pushback method [339] that collects packets in the data plane and send their digests to the control plane for further processing. A deep learning algorithm running in the control plane then identifies patterns and generates table entries to be pushed into the switch. This project was presented as a poster, and lacks implementation details.

Scholz et al. [225] presented a scheme that defends against SYN flood attacks through programmable switches. The authors considered the SYN authentication and SYN cookie defense strategies in their implementation. The idea of using programmable switches comes from the fact that it is computationally infeasible to defend against SYN flood attacks on the end hosts, specially with large volume of traffic. The system was implemented on a software switch (BMv2) as well as NetFPGA-SUME.

Zhang et al. [226] proposed Poseidon, a system that mitigates against volumetric DDoS attacks through programmable

TABLE XXIII
SUMMARY OF CYBERSECURITY LITERATURE.

Category	Ref	Name	Description	Implementation		
				Hardware (Tofino)	Software (BMv2)	Others
Heavy Hitters	[211]	HashPipe	Detects the k-th heavy hitter flows by maintaining counters		✓	
	[212]	N/A	Network-wide distributed heavy-hitter detection scheme	✓		
	[213]	Elastic Trie	Detects hierarchical heavy hitters using hashtable prefix tree		✓	
	[328]	PRECISION	Probabilistically recirculates a fraction of packets	✓		
	[214]	N/A	Monitoring distinct flows using HyperLogLog algorithm		✓	
Cryptography	[232]	N/A	Implementations of cryptographic hash functions			NetFPGA
	[233]	P4-IPsec	Implementation of host-to-site IPsec in P4 switches	✓		
	[234]	P4-MACsec	Implementation of MACsec on P4 switches		✓	
	[235]	N/A	AES implementation using scrambled lookup tables	✓		
Privacy and Anonymity	[229]	NetHide	Obfuscation of network topologies			N/A
	[228]	PANEL	In-network anonymity of the Internet forwarding infrastructure	✓		
	[230]	ONTAS	Anonymizes traffic online using P4 switches	✓		
	[231]	SPINE	Anonymizes traffic by concealing IP addresses and TCP fields		✓	
Access Control	[217]	Poise	Enforces access control to users based on contexts	✓		
	[218]	P40f	OS fingerprinting on programmable switches		✓	
Attacks Detection and Mitigation	[215]	P4Guard	Configurable firewall that acts based on predefined policies		✓	
	[216]	N/A	Delegates the authentication of end hosts to the data plane		✓	
	[219]	NETHCF	Hop-Count Filtering defense against spoofed IP traffic	✓		
	[220]	FastFlex	Abstraction that architects defenses into the network paths		✓	
	[221]	N/A	Discovers sensitivity attacks targeting data plane programs		✓	
	[222]	N/A	Distributed SIP DDoS defense mechanism		✓	
	[223]	N/A	Real-time DDoS attack detection using entropies		✓	
	[224]	ML-Pushback	Implements Pushback method on programmable switches			N/A
	[225]	N/A	Defends SYN flood attacks through programmable switches		✓	NetFPGA
	[226]	Poseidon	Mitigates against volumetric DDoS attacks	✓		
	[338]	N/A	Detection and mitigation of volumetric and stealth attacks		✓	
	[227]	NetWarden	Broad-spectrum defense against network covert channels	✓		

switches. It provides a language where operators can express a range of security policies. The defense primitives are partitioned by Poseidon to be executed on switches and on servers, based on their properties. The authors also developed a runtime management mechanism to adapt Poseidon's parameters in order to support dynamic defense without affecting legitimate flows. The system was implemented on a hardware switch (Tofino).

Friday et al. [338] proposed a unified in-network DDoS detection and mitigation strategy. The system considers both volumetric and slow/stealthy DDoS attacks. The system was validated against three use cases: UDP amplification, SYN flooding, and slow DDoS. The system was implemented on a software switch (BMv2), and results show the following: 1) with UDP, there is no degradation of legitimate users' QoS while UDP-based exploits take place; 2) TCP flooding were rendered ineffective and negated entirely; and 3) slow DDoS were detected, and services to authentic clients were re-established prior to one second of the attack reaching its peak, followed by a full mitigation within two seconds.

Xing et al. [227] proposed NetWarden, a broad-spectrum defense against network covert channels in a performance-preserving manner. NetWarden inspects and modifies headers for storage covert channel mitigation, without stalling the traffic. Regarding timing channel attacks, NetWarden uses data structures in the switch to monitor each connection and discover problematic protocol behaviors. The main motivation behind this system is that existing defenses are implemented in software, and therefore, are not able to perform per-packet operations at line rate. Moreover, existing schemes incur performance penalty for mitigation. The system was implemented on a hardware switch (Tofino) and the results show that the system is able to effectively mitigate covert timing and storage channels with minimal overhead.

E.3. Comparison between P4-based and traditional defense schemes

Network attacks such as large-scale DDoS and link flooding may have substantial impact on the network operation. For such attacks, server-based defenses deployed at the last mile are problematic and inherently insufficient, especially when attacks target the network core. Moreover, it is not feasible to detect and mitigate large volume of attack traffic (e.g., SYN flood) on end-hosts without impacting the throughput of the network. When defenses are architected into the network (i.e., detection and mitigation are programmed into the forwarding devices), it is easy to detect, throttle, or drop suspicious traffic at any vantage point, at line rate.

F. Summary

Table XXIII summarizes the works related to cybersecurity. In such area, there has been a wide range of works that aim at leveraging programmable switches to achieve the following goals: 1) detect heavy hitters and apply countermeasures; 2) execute cryptographic primitives in the data plane to enable further applications; 3) protect the identity and the behavior of end-hosts, as well as obfuscate the network topology; 4) enforce access control policies in the network while considering network dynamics; and 5) architect defenses in the data plane to accelerate the detection and mitigation processes.

XIV. Network Testing

Although programmable switches provide flexibility in defining the packet processing logic, they introduce potential risks of having erroneous and buggy programs. Such bugs may cause fatal damages, especially when they are unexpectedly triggered in production networks. In such scenarios, the network starts experiencing a degradation in performance as well as disruption in its operation. Bugs can occur in various phases in the P4 program development workflow (e.g., in the P4 program itself, in the controller updating data plane table entries, in the target compiler, etc.). Bugs are usually manifested after processing a sequence of packets with certain combinations not envisioned by the designer of the code. This section gives an overview of the troubleshooting and verification schemes for P4 programmable networks.

A. Troubleshooting

A.1. Background

Intensive research interests were drawn on troubleshooting the network. Previous efforts are mainly based on passive packet behavior tracking through the usage of monitoring technologies (e.g., NetSight [340], EverFlow [341]). Other techniques (e.g., Automatic test Packet Generation (ATPG) [342]) send probing packets to proactively detect network bugs. Such techniques have two main problems. First, the number of probe packets increases exponentially as the size of the network increases. Second, the coverage is limited by the number of probes-generating servers. Despite the flexibility that programmable switches offer, writing data plane programs increases the chance of introducing bugs into the network. Programs are inevitably prone to faults which could significantly compromise the performance of the network and incur high penalty costs.

A.2. Literature Review

Zhang et al. [236] proposed P4DB, an on-the-fly runtime debugging platform. The system debugs P4 programs in three levels of visibility by provisioning operator-friendly primitives: *watch*, *break*, and *next*. P4DB does not require modifying the implementation of the data plane. It was implemented and evaluated on a software switch (BMv2), and the results show that it is capable of troubleshooting runtime bugs with a small throughput penalty and little latency increase.

Zhou et al. [237] proposed P4Tester, a troubleshooting system for data plane runtime faults. It generates intermediate representation of P4 programs and table rules based on BDD data structure. Afterwards, it performs an automated analysis to generate probes. Probes are sent using source routing to achieve high rule coverage while maintaining low overheads. The system was prototyped for a software switch (BMv2) and a hardware switch (Tofino), and results show that it can check all rules efficiently and that the probes count is smaller than that of server-based probe injection systems (ATPG and Pronto).

Dumitru et al. [238] examined how three different targets, BMv2, P4-NetFPGA, and Barefoot's Tofino, behave when undesired behaviours are triggered. The authors first developed buggy programs in order to observe the actual behavior of targets. Then, they examined the most complex P4 program publicly available, switch.p4, and found that it can be exploited when attackers know the specifics of the implementation. In summary, the paper suggests that BMv2 leaks information from previous packets. This behavior is not observed with the other two targets. Furthermore, the authors were able to perform privilege escalation on switch.p4 due to a header destined to ensure communication between the CPU and the P4 data plane.

Kodeswaran et al. [239] proposed a data plane primitive for detecting and localizing bugs as they occur in real time. The authors used Ball-Larus encoding to profile the execution paths since it can encode all N paths of a program in a single $\log(N)$-bit variable. As P4 programs are loop-free and perform simple integer arithmetic operations, Ball-Larus encoding is a promising fit for tracking packet execution paths. The authors implemented a prototype where a P4 program is accepted as input, and an augmented P4 program which can track execution paths is generated. The system considered the switch.p4 program and was evaluated on a hardware switch (Tofino).

A.3. Comparison legacy vs. P4-based debugging

In legacy networks, network devices are equipped with fixed-function services that operate on standard protocols. Troubleshooting these networks often involve testing protocols and typical data plane functions (e.g., layer-3 routing) through rigid probing. On the other hand, with programmable networks, since operators have the flexibility of defining custom data plane functions and protocols, testing is more complex and is program-dependent. Probing-based approaches should craft patterns depending on the deployed P4 program. Other approaches proposed primitives that increase the levels of visibility when debugging P4 programs. Research work extracted from the literature show that it is essential to develop flexible mechanisms that operate dynamically on diverse P4 programs and targets.

B. Verification

B.1. Background

Program verification consists of tools and methods that ensure correctness of programs with respect to specifications and properties. Verification of P4 programs is an active area as bugs can cause faults that have drastic impacts on the performance and the security of networking systems. Static P4 verification handles programs before deployment to the network, and hence, cannot detect faults that occur at runtime. On the other hand, runtime verification uses passive measurements and proactive network testing. This section describes the major verification work pertaining to P4 programs.

B.2. Literature Review

Lopes et al. [240] proposed P4NOD, a tool that compiles P4 specifications to Datalog rules. The main motivation behind this work is that existing static checking tools (e.g., Header Space Analysis (HSA) [343], VeriFlow [344]) are

not capable of handling changes to forwarding behaviors without reprogramming tool internals. The authors introduced the "well formedness" bugs, a class of bugs arising due to the capabilities of modifying and adding headers.

Freire et al. [241, 242] proposed ASSERT-P4, a network verification technique that checks at compile-time the correctness and the security properties of P4 programs. ASSERT-P4 offers a language with which programmers express their intended properties with assertions. After annotating the program, a symbolic execution takes place with all the assertions being checked while the paths are tested. The system was implemented considering the reference P4 compiler (p4c), and evaluated by using applications from the literature.

Liu et al. [243] proposed p4v, a practical verification tool for P4. It allows the programmer to annotate the program with Hoare logic clauses in order to perform static verification. To improve scalability, the system suggests adding assumptions about the control plane and domain-specific optimizations. The control plane interface is manually written by the programmer and is not verified, which makes it error-prone and cumbersome. The authors evaluated p4v on both an open source and proprietary P4 programs (e.g., switch.p4) that have different sizes and complexities.

Nötzli et al. [244] proposed p4pktgen, a tool that automatically generates test cases for P4 programs using symbolic execution and concrete paths. The tool accepts as input a JSON representation of the P4 program (output of the p4c compiler for BMv2), and generates test cases. These test cases consist of packets, tables configurations, and expected paths. p4pktgen validates the test cases with a BMv2 software switch.

Lukács et al. [245] described a framework for verifying functional and non-functional requirement of protocols in P4. The system translates a P4 program in a versatile symbolic formula to analyze various performance costs. The proposed approach estimates the performance cost of a P4 program prior to its execution. The implementation is realized as a backend for the P4C compiler.

Stoenescu et al. [246] proposed Vera, a symbolic execution-based verification tool for P4 programs. The authors argue in this paper that a data plane program should be verified before deployment to ensure safe operations. Vera accepts as input a P4 program, and translates it to a network verification language, SEFL. It then relies on SymNet [345], a network static analysis tool based on symbolic execution to analyze the behavior of the resulting program. Essentially, Vera generates all possible packets layouts after inspecting the program's parser and assumes that the header fields can accept any value. Afterwards, it tracks the paths when processing these packets in the program following all branches to completion. For scalability improvements, Vera utilizes a novel match-forest data structure to optimize updates and verification time. Parsing/deparsing errors, invalid memory accesses, loops, among others, can be detected by Vera.

Shukla et al. [247] proposed P4RL, a reinforcement-learning fuzz testing system that automatically verifies P4 switches at runtime. The authors described a query language p4q in which operators express their intended switch behavior. A prototype that executes verification on layer-3 switch was implemented, and results show that PR4L detects various bugs and outperforms the baseline approach.

Dumitrescu et al. [248] proposed bf4, an end-to-end P4 program verification tool. It aims at guarantying that deployed P4 programs are bug-free. First, bf4 finds potential bugs at compile-time. Second, it automatically generates predicates that must be followed by the controller whenever a rule is to be inserted. Third, it proposes code changes if additional bugs remain reachable. bf4 executes a monitor at runtime that inspects the rules inserted by the controller and raises an exception whenever a predicate is not satisfied. The authors executed bf4 on various data plane programs and interesting bugs that were not detected in state-of-the-art approaches were discovered.

B.3. P4-based and traditional network verification

Traditional verification techniques that address the security properties in computer networks are mainly related to host reachability, isolation, blackholes, and loop-freedom. Techniques that check for the aforementioned properties include Anteater [346], which models the data plane as boolean functions to be used in a Boolean Satisfiability Problem (SAT) solver, NetPlumber [347] which uses header space algebra [343], and others (e.g., VeriFlow [344], DeltaNet [348], Flover [349], and VMN [350]).

Since P4 programs incorporate customized protocols and processing logic to be used in the data plane, traditional tools are not capable of handling changes to forwarding behaviors without reprogramming their internals. Therefore, verification techniques in programmable networks rely on analyzing the P4 programs themselves since they define the behavior of the data plane.

C. Summary

Table XXIV summarizes the works related to network testing. Network testing can generally be divided into debugging/troubleshooting network problems and verifying the behavior of forwarding devices. While traditional tools and techniques were adequate for non-programmable networks, they are insufficient for programmable ones due to their inability to handle changes to forwarding behaviors without reprogramming and restructuring their internals. A variety of works were proposed to analyze and model P4 programs in order to troubleshoot and verify the correctness of networks' operations.

XV. Distribution of the implementation platforms

Fig. 33 depicts the share of each implementation platform used in the surveyed papers, grouped by software, ASIC, NetFPGA, and SmartNICs. The graph shows that the vast majority of the works were implemented on software switches. Note that behavioral software switches (e.g., BMv2 [75]) are not suitable indicators of whether the program could run on a hardware target; they are typically used for prototyping ideas and to foster innovation. On the other hand, non-behavioral software switches (e.g., PICSES [48], derived from Open vSwitch (OVS) [351]) are production-grade and can be

TABLE XXIV
SUMMARY OF NETWORK TESTING LITERATURE.

Category	Ref	Name	Description	Implementation		
				Hardware (Tofino)	Software (BMv2)	Others
Troubleshooting	[236]	P4DB	On-the-fly runtime debugging platform		✓	
	[237]	P4Tester	Troubleshooting system for data plane runtime faults	✓		
	[238]	N/A	Examination of how different targets behave when undesired behaviors are triggered		✓	
	[239]	N/A	Ball-Larus encoding to profile the execution paths	✓		
Verification	[240]	P4NOD	Compiles P4 specifications to Datalog rules			✓
	[241, 242]	ASSERT-P4	checks at compile-time the correctness and the security properties		✓	
	[243]	p4v	Annotate program with Hoare logic clauses in static verification		✓	
	[244]	p4pktgen	Generates test cases using symbolic execution and concrete paths		✓	
	[245]	N/A	Analyze various performance costs of P4 program		✓	
	[246]	Vera	Symbolic execution-based verification tool for P4 programs			✓
	[247]	P4RL	Reinforcement-learning testing system that verifies P4 switches			✓
	[248]	bf4	Guarantees that deployed P4 programs are bug-free	✓		

deployed in data centers.
Hardware targets constitutes a smaller share of the platform distribution than software switches. A possible reasoning behind this is that the technology is still recent and targets are still not widely available for sale in the public. For example, to acquire a switch equipped with Tofino chip (e.g., Edgecore Wedge100BF-32 [20]), and to get the development environment and the customer support, the customer should sign a Non-Disclosure Agreement (NDA) with Barefoot Networks. Additionally, the customer should attend a training course (e.g., [352]) to understand the architecture and the specifics of the platform. This process is considered lengthy and costly, and not every institution is capable of affording it.

XVI. CHALLENGES AND CONSIDERATIONS

In this section, a number of research and operational challenges that correspond to the proposed taxonomy are outlined. The challenges are extracted after comprehensively reviewing and diving into each work in the described literature. Specifically, the challenges are divided into two main categories, namely, general challenges and application-specific challenges.

A. General Challenges

General challenges are independent of specific applications and should be considered when designing and developing any application on a programmable data plane switch.

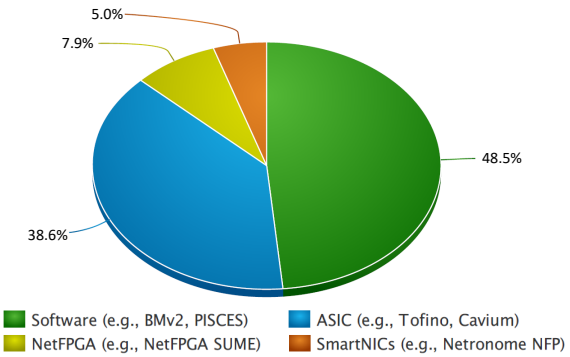


Fig. 33. Implementation platform distribution.

A.1. Memory availability (SRAM and TCAM)

The majority of innovative applications on the data plane benefit from the *stateful processing* property of the switch. Applications have the ability to store and manage information in data structures (e.g., arrays, bloom filters, sketches, etc.) on the data plane. Programmable hardware switches on the market are equipped with an on-chip SRAM that typically stores tens to hundreds of megabytes of data at most. Given the storage limitations of the switches, there are some practical concerns that must be addressed when implementing an in-network application. Below is a list of example storage concerns that correspond to specific applications.

(1) *Congestion control*: some CC schemes suggest dynamically rerouting traffic within the network when congestion occurs. Such schemes require storing alternative routes in the data plane. Other schemes require storing counters and/or flow states in memory to assist in controlling the congestion (e.g., throttling elephant flows). Since switches are constrained by the amount of memory they possess, the aforementioned CC schemes might not scale well for large topologies and networks.

(2) *Measurements and security*: the vast majority of measurement applications require storing metadata in the data plane (e.g., byte/packet counters). Additionally, the flows to be monitored are often stored in match-action tables with the help of the control plane. The number of flows to be measured and the richness of measurement information is bound by the size of the memory in the switch.

(3) *Machine learning*: It is not possible to store entire machine learning model updates on a switch at once. Therefore, ML systems that perform model aggregations in the data plane should be carefully designed to operate on smaller chunks of model updates (e.g., [197]).

(4) *Caching, AQMs, load balancing, telecom.*: keys/values, cache policies, state information, and load balancing information are all stored in the switch memory, limiting the scalability of the corresponding application.

Beside the size limitation of the on-chip memory, there are other restrictions that data plane developers should take into

account [353, 354]. First, since the table memory is local to each stage in the pipeline, other stages cannot reclaim non-utilized memory in other stages. As a result, memory and match/action processing are fuzed, making the placement of tables challenging. Second, the sequential execution of operations in the pipeline lead to poor utilization of resources especially when the matches and the actions are imbalanced (i.e., the presence of default actions that do not need a match).

A.2. Arithmetic computations.

There are several challenges when dealing with arithmetic computations in the data plane. First, programmable switches support a small set of simple arithmetic computations that operate on non-floating point values. Second, only few operations are supported per packet to guarantee the execution at line rate. Typically, a packet should only spend tens of nanoseconds in the processing pipeline. Third, computations in the data plane consume significant hardware resources, hampering the possibility of other programs to execute concurrently. Below is a list of example computations concerns that correspond to specific applications.

(1) *AQMs*: Some operations required by AQMs are complex to be implemented with P4 and can only be approximated. For example, the square root function, which is essential to the CoDel algorithm, is not supported in P4. Approximating this function can be implemented by counting the number of leading zeros through longest prefix match [153].

(2) *Machine Learning*: The majority of machine learning frameworks and models operate on floating point values while the supported arithmetic operations on the switch operate on integer values. For example, in-network model updates aggregation requires calculating the average over a set of floating-point vectors.

(3) *Hashing*: P4 currently supports non-cryptographic hash functions (e.g., CRC). These function can be easily implemented in embedded hardware and are computationally cheap when compared to cryptographic hash functions (e.g.,SHA-256). However, they do not satisfy the properties of strong cryptographic hash functions.

(4) *Cryptography*: encryption and decryption are expensive operations that are not currently supported natively by the data plane.

A.3. Network-wide Cooperation

The SDN architecture suggests using a centralized controller for network-wide switches management. Through centralization, the state of each programmable switch can be shared with other switches. Consequently, applications will have the ability to make better decisions as network-wide data is available locally on the switch. The problem with such architecture is the requirement of having a continuous exchange of packets with a software-based system. As an alternative, switches can exchange messages to synchronize their states in a decentralized manner.

A notable category of applications that heavily relies on the cooperation between switches is measurements and defenses. If switches do not cooperate, each will have its own local snapshot of the network state. Consequently, measurements tasks and in-network defenses will not have a network-wide view of traffic and events. For example, heavy hitters can go undetected if the traffic is monitored only at a single location [212]. The volume of traffic from a single sender on a given switch might not exceed a detection threshold. However, this threshold might be triggered if the total volume across all switches is considered. As a result, the cooperation of switches in measurements and security tasks considerably affect the system's accuracy and fidelity.

A.4. Control plane intervention

Delegating tasks to the control plane incurs latency and affects the application's performance. Ideally, this intervention should be minimized when possible. For example, to synchronize the state among switches, in-network cooperation should be considered.

In congestion control, rerouting-based schemes often use tables to store alternative routes. Since the data plane cannot directly modify tables, intervention from the control plane is required. The interaction with the control plane in this application hampers the promptness of rerouting.

Another example are methods that use collisions-free hashing which require intervention from the control plane. For example, cuckoo hash [355], which rearranges items to solve collisions, uses a complex search algorithm that cannot run on the switch ASIC, and is often executed on the switch CPU.

A.5. Security

When designing a system for the data plane, the developer must envision the kind of traffic a malicious user can initiate to corrupt the operation of the system. This class of attacks is referred to as *sensitivity attacks* as coined in [221]. Essentially, an attacker can intelligently craft traffic patterns to trigger unexpected behaviors of a system in the data plane. Below is a list of potential attacks that target specific applications.

(1) *Load balancing*: a load balancer that balances traffic by way of packet headers hashing without cyrptographic support (e.g., modulo operator on the number of available paths) can be tricked by an attacker that craft skewed traffic patterns. This results in traffic being forwarded to a single path, leading to congestion, link saturation, and denial of service.

(2) *Caching*: caching in data plane performs well when requests are mostly *reads* rather than *writes*. If an attacker continuously generates high-skewed write requests, the load on the storage servers would be imbalanced. If the system is designed to handle write queries on hot items in the switch, a random failure in the switch causes data to be lost.
An attacker can also exploit the memory limitation of switch and request diverse values, causing the pre-cached values to be evicted.

(3) *Congestion control*: network-assisted CC schemes provide explicit congestion feedback to end hosts. Consequently, the transmission rates of end-hosts are modified according to various network conditions. A man-in-the-middle attacker can exploit this mechanism and forge the congestion information in the packet to mislead the senders. For example, an at-

tacker can modify packets to force end-hosts to increase their transmission rates, leading to network congestion and service unavailability.

(4) *Multicast*: notable multicast works (e.g., [161]) encode the multicast tree in the packet header, as opposed to maintaining group-table entries inside routers. Additional security considerations must be taken into account when encoding multicast forwarding rules inside packets. For example, an attacker can forge the rules to flood or redirect traffic.

(5) *Telecommunications (media relay)*: the switch is used to relay traffic and bypass going through an x86-based server to improve latency, jitter, and throughput. An attacker can exploit this mechanism to turn the switch into a powerful relay of DDoS traffic.

(6) *Machine learning*: notable machine learning works (e.g., [197]) perform aggregation of model updates in the switch to overcome the overhead incurred when communicating with parameter servers. An infected parameter server can return invalid model updates vectors, leading to an incorrectly trained ML system.

(7) *In-network defenses*: detecting volumetric DDoS attacks in-network require comparing the traffic load against thresholds. If an attacker is aware about the detection mechanism and the utilized threshold, traffic can be meticulously crafted to remain below the radar (i.e., below the threshold).

B. Application-Specific Challenges

This section explores further challenges that are specific to different classes of P4 applications.

B.1. In-band Network Telemetry Variations

Many challenges and considerations are to be carefully considered when deploying INT in a network. Below is list of some important considerations that were extracted from the literature

(1) *Traffic overhead*: the vast majority of the INT variations works aim at minimizing the bandwidth overhead. INT generates additional telemetry packets to be sent to a remote monitor. In general, the bandwidth overhead is calculated as follows [117].

$$B = P \times (12 + N \times F \times 4) \times 8(bps), \tag{1}$$

where B is the bandwidth overhead, P is the number of INT packets per second, N is the number of traversed switches, and F is the number of monitored fields.

(2) *Packet overhead*: the packet size increases linearly with the path length as switches incrementally insert their telemetry data. Additionally, when further telemetry data is encoded in the headers, the payload ratio reduces in order to remain within the MTU.

(3) *Application performance*: embedding telemetry data increases packets' sizes. Consequently, the application suffers from increasing latency and jitter and degrading QoS. Moreover, the throughput of the application decreases as links are being occupied carrying INT packets.

(4) *Network coverage*: the scope of INT is limited in the sense that paths and metrics of interest have to be preassigned by operators and can not be altered at runtime. Therefore, only a small ratio of devices and links are tracked.

B.2. In-band Network Telemetry Collectors

The performance of collectors in an INT-enabled network is of utmost importance due to its impact on the richness of visibility of network events. The below challenges and considerations are vital for designing and deploying collectors for INT.

(1) *Collector resources*: the overhead of CPU and memory in collectors and monitoring engines decrease the packet processing rate. In particular, CPU usage and memory consumption increase gradually with the number of hops in the topology. Other characteristics that increase the load include number of metric values, activated INT fields, and frequency of network event.

(2) *Telemetry information archival*: Querying historical network information require high storage volume. Some existing INT collectors discard completely collected metrics (e.g., IntMon [122]).

B.3. Congestion Control

Network-assisted congestion control schemes should consider the following properties.

(1) *Deployability*: redesigning the whole infrastructure (e.g., switching behavior, routing, buffer requirements, transport protocol, etc.) when designing an in-network CC scheme might help in controlling and avoiding congestion. However, it is highly unlikely to be adopted by data centers and the Internet due to interoperability with contemporary infrastructure concerns. Furthermore, schemes that require modifying the end-hosts (software stack and/or hardware NICs) are not often not advised.

(2) *Topology dependence*: the effectiveness of congestion control in the data center depends heavily on the topology (e.g., fully-provisioned folded Clos topologies vs. asymmetric topologies (e.g., BCube, Jellyfish)). Ideally, an in-network CC scheme should generalize to various topologies.

(3) *Interoperability with other flows*: designing new CC schemes might negatively affect competing regular traffic. For example, NDP [126] shuts out competing TCP traffic if the same queue is used. Inteoperability with regular traffic (e.g., TCP) is essential to convince network operators to adopt a novel CC scheme.

(4) *Heuristics identification*: the type and the amount of traffic heavily depends on the network topology and the service provided. Therefore, different networks often witness distinct network conditions, especially when traffic is constantly varying. To this end, it is very challenging to identify thresholds and conditions that satisfy various network conditions.

B.4. Measurements

The following challenges and considerations pertaining to measurements in the data plane were extracted from the literature.

(1) *Generality and fidelity*: generality of monitoring tasks is preferred over crafting algorithms for specific tasks. This is achieved through *universal streaming*, a technique that allows a single universal sketch to be used for a variety of monitoring tasks. To be able to accommodate a wide range of tasks, the monitoring system should have no prior knowledge of the metrics, but still offer high accuracy.

(2) *Simplicity*: some monitoring systems allow network operators to express the monitoring tasks through a query-based language. Such language should be relatively straightforward so that operators can innovate when defining monitoring tasks.

(3) *Tradeoffs*: with respect to measurements in the data plane, there are tradeoffs between speed, accuracy, scalability, and capabilities. The program designers should prioritize their requirements and tune the monitoring system accordingly.

B.5. AQMs

The following considerations and challenges were extracted for AQMs.

(1) *Queueing information availability*: Some AQMs (e.g., PIE) suggest dropping some packets before enqueueing them in the ingress pipeline. In P4, the ingress pipeline has no information pertaining to current queueing delay.

(2) *Fixed-function traffic managers*: most hardware switches are equipped with non-programmable traffic managers (e.g., Packet Replication Engine and Traffic Manager (PRE/TM) in Tofino-based devices). Consequently, it is not possible for a P4 program to access and modify parameters. For example, recent congestion control algorithms (e.g., BBR [356]) favor small buffers. Changing the buffer size is only possible from the control plane, and hence, it is not possible to dynamically adjust it through the P4 program based on the queueing conditions.

B.6. QoS/TM

The following considerations and challenges were extracted for QoS and TM.

(1) *QoS information in application layer*: improving QoS by inspecting application-layer headers is not always feasible, especially when data is encoded in variable-length fields and is encoded deep in the packet. This a limitation of both the language and the hardware. P4 language lacks loops support, and the headers and metadata bus are limited in size.

(2) *Generality*: different capabilities for metering traffic are provided by different switch vendors; therefore, some designs cannot be generalized.

(3) *Adaptability*: Traffic management should be adaptive with churned workloads. Moreover, it should have the ability to be changed at runtime depending on traffic load changes.

(4) *Transparency*: traffic management should operate without modifying existing user stack, protocols, applications, and/or operating systems.

B.7. Multicast

The main challenge/consideration extracted for multicast is interoperability with legacy devices. Programmable switches provide scalability benefits when multicast forwarding rules are encoded inside packets (e.g., ELMO [161]). However, when a network is equipped with both legacy and programmable switches (e.g., for incremental deployment purposes), the group-table sizes on legacy devices will remain a bottleneck.

B.8. Load Balancing

Below is a list of challenges and considerations for load balancing schemes.

(1) *Link failures*: the load balancing algorithm should be able to adapt to link changes (e.g., link goes down). For example, when a broken link comes up again, the algorithm should stabilize and redistribute the load and the utilization values on all links.

(2) *Handle switch failures*: when a switch goes down, the system should automatically redirect connections to other operating switches, as long as all switches maintain the same load balancing table.

(3) *Handle DIP failures*: the load balancing algorithm should detect whenever a DIP failed (i.e., server from the pool is unreachable) and should evict its entry from the load balancing table.

(4) *Hybrid operation*: when the resources on the switch are full (i.e., table is exhausted), the system should consider software-based load balancers as an alternative option for newly incoming connections.

B.9. Caching

The following considerations and challenges were extracted for caching schemes.

(1) *Communication overhead*: communicating with the storage servers to retrieve values is expensive.

(2) *Keys/values freshness management*: detecting and managing keys/values to be cached is challenging, especially when there are frequent differences in the distribution of the requests.

(3) *Multi-rack caching*: when caching systems are deployed on ToR switches, they operate on a single rack (e.g., NetCache). When the number of racks increases, load imbalance on the individual racks might become an issue.

(4) *Key/values length*: the length of the values to be cached (keys and values) is not flexible. Specifically, it is limited to the length of the packet headers.

(5) *Write-intensive workloads*: in-network caching works best with read-intensive workloads. However, when the write requests dominate, storage server must process these requests. Consequently, write queries would be lost under switch failures.

(6) *Access patterns*: when the cached items (keys and values) contain critical information, they are typically encrypted to protect their confidentiality. Even though in-network caching systems are able to operate on encrypted keys, the key access patterns are always revealed.

B.10. Telecommunication services

The following considerations and challenges were extracted with respect to telecommunications services accelerated by the data plane.

(1) *Generality*: the selection of values to be included in the matching rules and the action data while accommodating a wide range of use cases in telecommunications is a challenge.

(2) *Signalling offloading*: offloading signalling protocols (e.g., SIP) to the data plane is difficult due to the complexity of the protocol logic in the application-layer.

(3) *Scalability*: network elements (e.g., vEPG) encompass a variety of services (e.g., charging, rate limiting, etc.). There is a trade-off between the scalability (number of active users) and the concurrent supported services.

B.11. Publish/Subscribe

The challenges and considerations of publish/subscribe schemes are as follows.

(1) *Reliability*: maintaining a reliable and persistent communication which is typical in contemporary pub/sub systems is challenging and requires additional processing.

(2) *Message retaining*: message retaining, which is a mechanism that stores and distributes previous messages to recently subscribed users, requires tracking active users and storing historical messages in the data plane.

(3) *QoS differentiation*: the guarantee of delivery for a specific message cane be specified in legacy pub/sub systems (e.g., Message Queueing Telemetry Transport (MQTT)). For instance, a client can specify whether a message should be receieved at most once (QoS = 0), at least once (QoS = 1), or exactly once (QoS = 2). Implementing this mechanism in the data plane is challenging.

B.12. Consensus

The following list describes the challenges that were extracted when reviewing the literature regarding consensus algorithms.

(1) *Replicated items length*: the values to be replicated and on which consensus runs is limited to the maximum transmission unit.

(2) *Variable length values*: it is complicated to support variable-length values due to the lack of loops and iteration in P4.

(3) *System failures*: the consensus algorithm should anticipate failures in the components (e.g., leader, acceptor, etc.) and should not prevent the connectivity between the participants.

B.13. Machine Learning

With respect to machine learning, the following notions are to be considered.

(1) *In-network inference*: performing machine learning inference in the data plane is challenging due to the limited capabilities of switch with respect to computations.

(2) *Encrypted traffic*: in-network aggregation is not straightforward when the traffic is encrypted.

(3) *Handling losses*: packet loss prevents the switch from completing the aggregation process. Therefore, a loss recovery mechanism should be implemented when aggregation is performed in the network.

B.14. IoT Aggregation/Disaggregation

With respect to in-network packets aggregation and disaggregation, the following challenges and considerations are extracted.

(1) *Disaggregation overhead*: although the achieved throughput in the aggregation process matches the maximum capacity of a switch port, the disaggregation throughput is constrained by the bandwidth of internal I/O buses in the used P4 chip.

(2) *Packet size variability*: IoT packets might have different sizes and variable-length fields. Such characteristics necessitate additional parsing and processing logic on the switch.

(3) *Idle state management*: when the traffic is low and the network is idle, there might not be enough packets to be aggregated. This phenomenon causes excessive delays and affects the application performance.

B.15. IoT Service Automation

The challenges and considerations pertaining to service automation are summarized as follows.

(1) *Non-standard IP protocols*: supporting non-IP protocols translation require developing translators for each protocol. Such approach may not be feasible when many protocols (e.g., as in smart cities) exist in the network.

(2) *Non-Ethernet protocols*: available hardware switches in the market are Ethernet-based; therefore, non-Ethernet approaches are limited to software switches (e.g., PISCES).

B.16. Heavy Hitter

The main challenge with heavy hitters is the detection accuracy. False positives (i.e., reporting a normal flow as a heavy hitter) and false negatives (i.e., not reporting a heavy hitter flow) could have dramatic impact on the network, depending on the application. For instance, if a heavy hitter is treated as a DDoS attack initiator, a high false positives rate will overwhelm the operators with false alarms. At the same time, a high false negative rate will disrupt the network operation and render the network unavailable.

B.17. Privacy and Anonymity

With respect to privacy and anonymity, the following considerations are noted.

(1) *Tracing practicality*: when obfuscating the topology, the practicality of path tracing tools should be preserved.

(2) *Robustness*: the obfuscation algorithm should be strong enough not to be inverted.

(3) *Failure detection*: link failures in the physical topology should remain visible after obfuscation.

(4) *Compatibility with legacy*: when randomizing identifiers to achieve session unlinkability, the identifiers must fit into the small fixed header space so that compatibility with legacy networks is preserved.

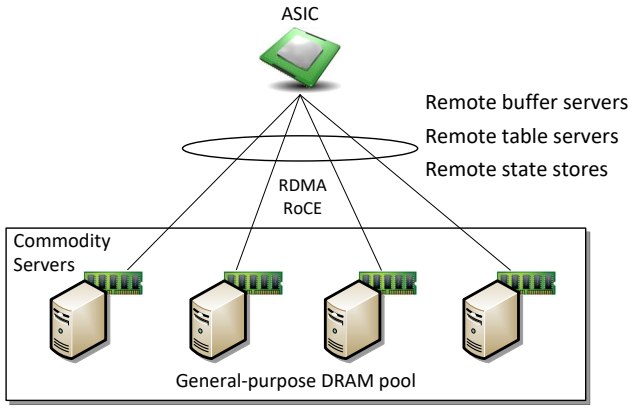


Fig. 34. Expanding switch memory by leveraging remote DRAM on commodity servers [357].

(5) *Hash collisions*: anonymization schemes that rely on hashing are limited to use non cryptographic hashing available in P4 (e.g., CRC).

B.18. Access Control

The following considerations are related to access control schemes in the data plane.

(1) *Authentication*: authorization and access control schemes require authentication prior to authorization. Poise [217] for instance controls access to enterprise resources only after users are authenticated.

(2) *Cryptographic hashing*: access control schemes often require the usage of cryptographic primitives (e.g., [217]) which is not natively available in P4.

(3) *Switch placement*: access control at the edge is preferred over last mile authentication to avoid overwhelming the network with unauthorized requests.

B.19. In-network Defenses

When architecting defenses in the data plane, the following considerations and challenges must be taken into account.

(1) *Hardware resources management*: when integrating security functions alongside routing and switching, hardware resources such as memory, hardware stages, and ALUs should be multiplexed.

(2) *Switch placement*: the placement of defenses in the network affects flow visibility and hence has an impact on the network-wide attacks detection. For example, the detection and mitigation of attacks at the last mile can be insufficient, especially if the attack targets the network core (e.g., link flooding attack). Finding the optimal placement is challenging due to unpredictable attack patterns.

XVII. Discussions and Future Perspectives

This section discusses and pinpoints several initiatives for future work which could be worthy of being pursued in this imperative field of programmable switches.

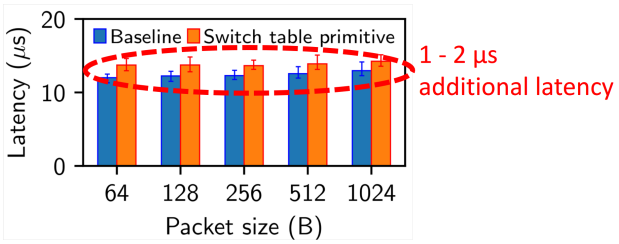


Fig. 35. Accessing remote DRAM latency overhead. Achieved throughput close to the line rate (≈ 37.5 Gbps) [357].

A. Application-Agnostic Trends

A.1. Memory Expansion and Disaggregation

Stateful processing is a key enabler for programmable data planes as it allows applications to store and retrieve data across different packets. This advantage enabled a wide range of novel applications (e.g., in-network caching, fine grained measurements, stateful load balancing, etc.) that were not possible in non-programmable networks. The amount of data stored in the switch is limited by the size of the on-chip memory which ranges from tens to hundreds of megabytes at most. Consequently, the majority of stateful-based applications suffer have trade-offs between performance and memory usage. For instance, the efficiency of caching which is determined by the *hit rate* is directly affected by the memory size.

A notable work by Kim et al. [357, 358] suggests accessing remote Dynamic Random Access Memory (DRAM) installed on data center servers purely from data plane to expand the available memory on the switch. The bandwidth of the chip is traded for the bandwidth needed to access the external DRAM. The authors analyzed traffic traces from production clusters (Facebook, Alibaba). First, they found that that servers' DRAMs are mostly underutilized (median usage ≈ 50%, 99th percentile ≈ 80%). Second, network link bandwidth in data center networks is also underutilized (average link utilization between ToR switch and server is ≈ 1%, and ≈ 10% on heavy loads). These underutilized resources are leveraged to expand the memory capacity of the programmable switches. The approach is cheap and flexible since it reuses existing resources in commodity hardware without adding additional infrastructure costs. Essentially, this work introduces the memory hierarchy concept to programmables switches. The on-chip SRAM represents the cache and the remote DRAM represents the main memory, similar to the memory hierarchy in CPU architectures. The system is realized by allowing the data plane to access remote memory through an access channel (RDMA over Converged Ethernet (RoCE)) as shown in Fig. 34. This work considered three use cases: 1) remote packet buffer; 2) remote lookup table; and 3) remote state stores. The idea was evaluated on a programmable switch with Tofino ASIC and a server equipped with a 40Gbps Mellanox CX-3 Pro NIC. Results show that 1) the packet buffer was increased by 1000× (from O(10 MB) to O(10GB)); 2) the remote lookup table with exact matching was also increased by 1000×; and 3) up to 95.6% RDMA NIC bandwidth with 1-2 microseconds extra latency (Fig. 35) and zero CPU overhead.

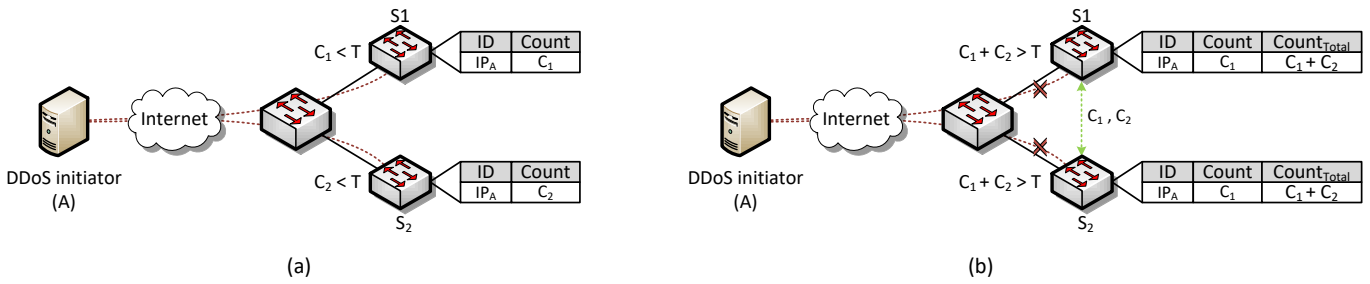


Fig. 36. (a) Local detection of DDoS attacks. (b) network-wide detection of DDoS attacks.

There are some limitations in the aforementioned approach that can be explored in the future. These future directions will be beneficial for the vast majority of stateful data plane applications.

- The current implementation only supports address-based memory access, and hence, complicated data layouts and ternary matching in remote memory should be explored.
- Frequent updates in the remote memory requires several packets for fetching and adding. This is common in measurement applications where counters are continuously incremented. A possible solution to the bandwidth overhead is aggregating updates into single operation. This comes with the cost of having delays in the updates.
- Packet loss between the switch and the remote memory should be handled, otherwise, the performance of the application and the freshness of the remote values might be affected.
- The interaction between general data plane applications and the remote memory is challenging. A potential improvement is designing well-defined APIs to facilitate the interaction.

Another notable work by Chole et at. [354] explored the idea of disaggregating the memory and compute resources of a programmable switch. The main notion of this work is to centralize the memory as a pool that is accessed by a crossbar. By doing so, each pipeline stage no longer has local memory. Additionally, this work solves the sequential execution limitation by creating a cluster of processors used to execute operations in any order.

A.2. In-network Computing on Demand

Based on the results of numerous works discussed in this survey, programmable data planes proved to enhance the performance of various applications by orders of magnitude. Despite these improvements, operators are hesitant to deploy programmable switches. This reluctance comes from the conventional wisdom that hardware acceleration is power hungry, and hence increases operational costs. Tokusashi et al. [359] analyzed this assumption by comparing the power consumption of programmable switches running different data plane programs to that of x86-based servers running the same programs. Results show that 1) the power consumption increase in programmable switches over non-programmable switches is small; 2) when idle, the server-based systems are more power efficient. However, as processing exceeds 10% of the CPU capacity, the power consumption increases substantially; 3) the workload does not affect the power consumption of programmable switches.

These results motivated the idea of in-network computing on demand [359]. Essentially, this idea suggests shifting the computations between servers and switches, as long as both run identical applications. When the energy of the system running on the server exceeds the energy of the network, the computation is shifted to the network.

There are some challenges in this approach that can be explored in the future. For example, a potential work could be the ability to shift any type of application to the network. Another idea to explore is the scheduling of in-network computing on demand in data centers based on power consumption. Finally, the reliability of in-network computing on demand should be addressed.

A.3. Switch State Synchronization

Network-wide collaboration to synchronize the state of switches can improve the accuracy of a given system.

Consider Fig. 36 which shows an in-network DDoS defense solution. Each switch maintains a list of senders and their corresponding numbers of bytes. A switch compares the number of bytes transmitted from a given flow to a threshold. When the threshold is crossed, the flow is blocked and the device is identified as a malicious DDoS sender. Assume that the network implements a load balancing mechanism that distributes traffic across the switches. In the scenario where switches do not consider the byte counts of other switches (Fig. 36 (a)), the traffic of a DDoS device might remain under the threshold. On the other hand, when switches synchronize their states by sharing the byte counts (Fig. 36 (b)), the total number of bytes is compared against the threshold. Consequently, the total load of a DDoS device is considered. This example demonstrates an application that heavily depends on network-wide cooperation and hence motivates the need for state synchronization.

The SDN architecture which suggests using a centralized controller to synchronize the states among switches is not feasible with programmable data planes. The reason for this infeasibility is that states in the data plane are updated at line rate, possibly with speeds that can exceed Tbps rates. Consequently, software-based controllers cannot keep up with these speeds, which leads to network-wide states inconsistencies. Moreover, the controller might not be aware of the states to be migrated since they are different for each device at runtime (e.g., a register array indexed with the hash of

packets' headers). a variety of works aim at solving the state synchronization problem in the data plane.

Arashloo et al. [360] proposed SNAP, a centralized stateful programming model that aims at solving the synchronization problem. SNAP introduced the idea of writing programs for "one big switch" instead of many. Essentially, developers write stateful applications without caring about the distribution, placement, and optimization of access to resources. SNAP's compiler automatically maps everything based on the reads and writes dependencies in the code. SNAP is limited to one replica of each state in the network. Sviridov et al. [361, 362] proposed LODGE and LOADER to extend SNAP and enable multiple replicas. Luo et al. [363] proposed Swing State, a framework for runtime state migration and management. This approach leverages existing traffic to piggyback state updates between cooperating switches. Swing State overcomes the challenges of the SDN-based architecture by synchronizing the states entirely in the data plane, at line rate, and without intervention from the control plane. There are several limitations with this approach. First, there are no message delivery guarantees (i.e., packets dropped/reordered are not retransmitted), leading to inconsistency in the states among the switches. Second, it does not merge the states if two switches share common states. Third, the overhead can significantly increase if a single state is mirrored several times. Finally, there is no authentication of data or senders. Xing et al. [364] proposed P4Sync, a system that migrates states between switches in the data plane while guaranteeing the authenticity of the senders and the exchanged data. P4Sync addresses the limitations of existing approaches. It guarantees the completeness of the migration, ensuring that the snapshot transfer is completed. Moreover, it solves the overhead of the repeatedly retransmitted updates. An interesting aspect of P4Sync is its ability to control the migration traffic rate depending on the changing network conditions.

The future work is this area should consider handling *frequent state migrations*. Some systems require migration packets to be generated each RTT, causing increased traffic overhead and additional expensive authentication operations. For instance, P4Sync uses public key cryptography in the control plane to sign and verify the end of the migration sequence chain (2.15ms for signing and 0.07ms to verify using RSA-2048 signature). Frequent migrations would cause this signature to be involved repeatedly. Another major concern that should be handled in future work is *denial of service*. Even with migration updates authentication, changes in the packets cause the receiver to reject updates, leading to state inconsistency among switches.

B. Application-specific Trends

B.1. INT Variations

The below list outlines current and potential future directions with respect to INT variations.

(1) *Header compression*: compressing INT headers size to preserve bandwidth and improve Jitter.

(2) *Reporting frequency tuning*: Ideally, INT solutions should not impact the application QoS (e.g., latency, jitter). An intelligent reporting algorithm decreases reporting frequency without sacrificing the accuracy. It also improves the application throughput by controlling the excessive number of INT reports.

(3) *Coverage broadening*: increasing coverage to gain a comprehensive view of the network.

(4) *Information richness*: the INT specification defines a limited set of telemetry metadata which is supported by most switches. Due to the flexibility of programming the data plane, additional measurement information could be embedded in the telemetry reports.

(5) *Simplified deployment*: Simplifying the deployment and minimizing interactions between INT-capable devices while maintaining a broad coverage of the network.

(6) *Monitoring policies*: developing high-level languages that translate monitoring policies into individual devices programs. A translator could use optimization techniques while assigning tasks for each device to regulate the traffic overhead.

B.2. INT Collectors

Potential research directions for INT collectors include the following.

(1) *Collector resource management*: enhance the performance of INT collectors using packet processing accelerators (e.g., DPDK, SmartNICs) to accommodate higher rates while maintaining tolerable CPU and memory overhead. Additionally, explore RDMA-based approaches that leverage remote DRAM on collectors to minimize the reporting delay.

(2) *Telemetry archival*: efficiently store the telemetry data in structured file systems and databases to enable operators to query historic telemetry trends of the network.

B.3. Congestion Control

(1) *Transparent deployment*: control and avoid congestion in networks without modifying the existing infrastructure (switching and routing devices, end-hosts).

(2) *Topology-agnostic integration*: generalize CC schemes to any topology and traffic workload in the network.

(3) *Fairness improvement*: stabilize the performance of regular traffic (e.g., TCP) when competing with newly proposed network-assisted CC protocols.

(4) *Parameters minimization*: Develop parameterless CC schemes in order to scale and generalize the deployment.

(5) *State storage reduction*: minimize the storage of flows states and statistics in the network to improve scalability.

(6) *Security consideration*: network-assisted CC schemes should take security into consideration, especially when explicit congestion feedback (e.g., HPCC [128]) is provided by the switches.

(7) *Dynamic buffer tuning*: the rule-of-thumb packet buffer size is one bandwidth-delay product (BDP). Recent CC protocols (e.g., BBR) favor buffers that are smaller than 1BDP [365]. A potential research project is to dynamically modify the packet buffer size after inspecting flows' characteristics (e.g., CC algorithm, heavy hitter, MSS, etc.).

B.4. Measurements

(1) *Storage reduction*: perform network-wide measurements tasks with minimal storage on the switch.

(2) *Generalized monitoring*: generic measurements platform that is customizable is preferred for scalability.

(3) *Simplified configuration*: demystify and minimize the interactions between the operators and the measurement system when configuring monitoring tasks.

(4) *Tradeoffs management*: maximization of performance, accuracy, scalability, and monitoring capabilities. The monitoring system should provide the operator the flexibility to tune parameters so that a balance between the tradeoffs is achieved.

(5) *Monitoring primitives extension*: extend the capabilities of the data plane to support new aggregations and computations on network traffic (e.g., link utilization microbursts).

(6) *Controller computations avoidance*: minimize (or possibly eliminate) computations on real-time traffic in the controller.

(7) *Placement optimization*: infer optimal switch placement while maximizing flow visibility and interworking with legacy devices.

B.5. AQMs

(1) *Novel AQMs development*: develop new AQMs and experiment with ideas that contribute to the bufferbloat problem, considering switch metadata (e.g., telemetry information) and the programmability of the switch.

(2) *Existing AQMs implementation*: implement existing AQMs and their variations on P4 as open source. Having those codes publicly available would facilitate the development on specific targets (e.g. Tofino) for interested operators.

(3) *Complex operations implementation*: improving approximations of mathematical computations in the data plane to support complex operations.

(4) *Intelligent control plane interaction*: delegate computations that are not executed at line rate for each packet to the control plane.

(5) *Thresholds adaptability*: dynamically adapt AQMs' thresholds based on measurements performed in the data plane to cope better with diverse traffic natures.

(6) *Advanced data structures development*: use advanced data structures to track flows if needed with minimal storage (e.g., sketches).

B.6. QoS and TM

(1) *Packet QoS inspection*: explore approaches that process QoS information embedded in higher layers at line rate speeds.

(2) *QoS information compression*: systems that embed QoS in packets should consider compressing the information without losing richness.

(3) *Control plane interaction minimization*: intelligently applying traffic policies and algorithms entirely in the data plane without intervention from the control plane.

(4) *Metering capabilities generalization*: unifying metering capabilities and algorithms across various targets.

B.7. Multicast

(1) *Incremental deployment*: explore strategies to deploy multicast-based approaches incrementally while coexisting with legacy switches.

(2) *Reliability enhancement*: native multicast works on a best-effort basis. To improve reliable delivery, multicast solutions should consider layering protocols such as PGM and SRM.

(3) *Multicast monitoring*: for troubleshooting purposes, systems can explore INT and other telemetry approaches for monitoring multicast packets.

B.8. Load Balancing

(1) *Sacalability improvement*: utilize compact data structures and develop algorithms that do not require a large amount storage in the switch.

(2) *Hashing-based algorithms improvement*: develop approaches that produce collision-free hashes entirely in the data plane.

(3) *Flow priority-based approaches*: develop systems that consider flow priority in the load balancing algorithm.

(4) *Multipath transport protocols consideration*: develop approaches for recent multipath transport protocols such as Multipath QUIC.

(5) *Traffic classification*: develop systems that provide the operator the freedom in specifying traffic to be splitted.

B.9. Caching

(1) *Information compression*: compress keys and values to be cached to decrease the SRAM usage.

(2) *Communication minimization*: create algorithms that retrieve values from storage servers with minimum communication overhead.

(3) *Priority-based caching*: enable operators to provide priorities to keys/values to be stored. Such system enforce a cached packet not be evicted when the SRAM fills up.

(4) *Values aggregation caching*: systems can consider caching the aggregations of frequent values (e.g, averages).

B.10. Telecommunication Services

(1) *Scalability improvement*: support multiple services and telecom applications while accommodating a large number of active users.

(2) *Signalling offloading*: designing non-complex signalling protocols that can be easily processed by the data plane.

(3) *In-network media mixing*: explore the possibility of mixing media streams in the data plane (e.g., voice and video conferencing).

B.11. Publish/Subscribe

(1) *Reliability insurance*: create publish/subscribe systems that provide a reliable communication and guarantee packet delivery.

(2) *Message retaining*: develop publish/subscribe approaches

that temporarily retain messages in the data plane. When a timer expires, the retained messages are managed by the control plane.

(3) *QoS differentiation consideration*: enable QoS differentiation in in-network publish/subscribe systems. A possible approach is to develop QoS with the same semantics as those used in MQTT.

B.12. Consensus

(1) *Novel consensus algorithms development*: design novel consensus algorithms that diverge from the existing algorithms (e.g., Paxos, Raft) and exploit the data plane programmability to contribute to state-of-the-art systems.

(2) *Failure recovery mechanisms*: develop prompt failure recovery mechanisms for consensus algorithms that maintain connectivity between participants.

(3) *Decoupled implementation*: decouple the functionality of complex consensus algorithms between the data plane and the control plane. Additionally, provide correctness proofs that the proposed system achieves the same (or better) results than contemporary algorithms.

B.13. Machine Learning

(1) *In-network inference systems development*: design and develop machine learning systems that performs inference in the network. The complexity of the system can be tackled through a decoupled architecture where tasks are distributed between the data plane and the control plane.

(2) *Loss recovery mechanisms*: create recovery mechanisms for in-network aggregation to overcome packet loss.

(3) *Feature extraction acceleration*: develop machine learning models that leverage programmable switches to extract features. This is advantageous due to the fact that it is possible to operate on traffic with high rates in the data plane.

B.14. Aggregation/Disaggregation

(1) *Packet performance analysis*: study the performance side effects (e.g., delay, jitter, loss rate, retransmission) that aggregation causes to packets.

(2) *Idle state management*: timers should be implemented to avoid excessive delays resulting from waiting for enough packets to be aggregated.

B.15. IoT Service Automation

(1) *Protocol translators development*: design and develop translators for non-IP IoT protocols so that applications on various devices that run different protocols can exchange data.

(2) *Non-Ethernet approaches exploration*: leverage production-grade software switches to support non-Ethernet IoT protocols.

B.16. Heavy Hitters

(1) *Advanced data structures*: explore more complex data structures to reduce the amount of state storage required on the switches.

(2) *Heavy distinct count*: commutative and associative techniques are not effective when computing distinct counts. Approximation algorithms can be explored for heavy distinct count.

(3) *Accuracy improvement*: develop heavy hitter detection mechanisms that minimize false positives and false negatives of state-of-the-art systems.

(4) *Incremental deployment strategies*: explore strategies for incremental deployment while maximizing visibility.

B.17. Privacy and Anonymity

(1) *Incremental deployment strategies*: explore strategies for incremental deployment while maximizing visibility.

(2) *Session setup latency minimization*: session initiation in anonymization solutions often requires interaction with the CPU/Controller (e.g., [228]). Future work should explore strategies to minimize the session setup time.

(3) *Anonymization based on encryption*: header and payload encryption can improve the anonymity of the session. Future work should consider exploring methods to leverage encryption while maintaining line rate.

B.18. Access Control

(1) *Authentication schemes*: Future work should explore in-network methods to authenticate the users.

(2) *Next-generation firewalls*: explore the next generation firewall functionalities that can be offloaded to the data plane. For instance, in [202], the authors proposed a system that allows searching for keywords in the payload of the packet. Similar techniques could be leveraged to imitate the functionalities of a next-generation firewall.

(3) *Cryptographic primitives support*: supporting cryptographic primitives in the data plane can tremendously help in access control (e.g., securing context packets in Poise [217]).

C. In-Network Defenses

(1) *Traffic characterisation*: attacks detection (e.g., ransomware detection) is heavily based on characterizing the traffic. Future work should consider characterizing network traffic under various conditions, and considering different deployments, to propose in-network mitigation best practices.

(2) *Sensitivity attack patterns characterization*: sensitivity attacks consist of crafting traffic patterns that trigger unexpected behaviors in the data plane. It is worth exposing sensitivity attack patterns for different application types so that data plane developers can avoid the vulnerabilities that trigger those attacks in their codes.

(3) *Stealthy DDoS mitigation*: the majority of the works in the literature mitigate volumetric DDoS attacks. Future work should consider slow DDoS (also known as stealthy and low volume attacks) mitigation techniques.

XVIII. Conclusions

This article presents a tutorial on P4 language and an exhaustive survey an programmable data planes. The survey

describes the evolution of networking by discussing the traditional control plane and the transition to SDN. Afterwards, the survey motivates the need for programming the data plane and delves into the general architecture of a programmable switch (PISA). A brief tutorial on P4, the de-facto language for programming the data plane was presented, explaining the building blocks of the language and a sample program achieving IP forwarding. Motivated by the increasing trend in programming the data plane, the survey provides a taxonomy that sheds the light on numerous significant works and compares each category in the taxonomy with that in legacy approaches. The survey concludes by discussing challenges and considerations as well as various potential future work.

ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under grant numbers 1925484 and 1829698, funded by the Office of Advanced Cyberinfrastructure (OAC).

TABLE XXV
ABBREVIATIONS USED IN THIS ARTICLE.

Abbreviation	Term
ABR	Adaptive Bit Rate
ACK	Acknowledgement
ACL	Access Control List
ACM	Association for Computer Machinery
AFQ	Approximate Fair Queueing
AIMD	Additive Increase Multiplicative Decrease
ALU	Arithmetic Logical Unit
AMD	Advanced Micro Devices
API	Application Programming Interface
AQM	Active Queue Management
AS	Autonomous System
ASIC	Application-specific Integrated Circuit
ATPG	Automatic Test Packet Generation
ATT	Attribute Protocol
BBR	Bottleneck Bandwidth and Round-trip Time
BDD	Binary Decision Diagram
BFT	Byzantine Fault Tolerance
BGP	Border Gateway Protocol
BIER	Bit Index Explicit Replication
BLE	Bluetooth Low Energy
BLESS	Bluetooth Low Energy Service Switch
BMv2	Behavioral Model Version 2
BNN	Binary Neural Network
BQE	Buffer Queueing Engine
BQPS	Billion Queries Per Second
BYOD	Bring Your Own Device
CAIDA	Center Applied Internet Data Analysis
CC	Congestion Control
CLI	Command-Line Interface
CNN	Convolutional Neural Network
CoDel	Controlled Delay
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CU	Central Unit
CWND	Congestion Window
DCQCN	Data Center Quantized Congestion Notification
DCTCP	Data Center Transmission Control Protocol
DDoS	Distributed Denial of Service
DHCP	Dynamic Host Configuration Protocol
DIP	Direct Internet Protocol
DMA	Direct Memory Access
DMZ	Demilitarized Zone
DNS	Domain Name Server
DPDK	Data Plane Development Kit
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processors
ECMP	Equal-Cost Multi-Path Routing
ECN	Explicit Congestion Notification
EIGRP	Enhanced Interior Routing Protocol
EPG	Evolved Packet Gateway
ESP	Encapsulating Security Payload
FAST	Flow-level State Transitions
FBOSS	Facebook Open Switching System
FCT	Flow Completion Time
FIB	Forwarding Information Base
FPGA	Field-programmable Gate Array
FQ	Fair Queueing
FRR	Free Range Routing
GPU	Graphics Processing Unit
GRE	Generic Routing Encapsulation
HCF	Hop-Count Filtering
HPCC	High Precision Congestion Control
HSA	Header Space Analysis
HTCP	Hamilton Transmission Control Protocol
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IGMP	Internet Group Management Protocol
IKE	Internet Key Exchange
ILP	Integer Linear Programming
INT	In-band Network Telemetry
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
IT	Information Technology
JSON	JavaScript Object Notation
KDN	Knowledge-defined Networking
KPI	Key Performance Indicator
LAN	Local Area Network
LFA	Link Flooding Attack
LPM	Longest Prefix Match
LPWAN	Low Power Wide Area Network
LTE	Long Term Evolution
MAC	Medium Access Control
MAU	Match-Action Unit
MCM	Multicolor Markers
MIMD	Multiplicative Increase Multiplicative Decrease
ML	Machine Learning
MOS	Mean Opinion Score
MPC	Mobile Packet Core
MPLS	Multiprotocol Label Switching
MQTT	Message Queueing Telemetry Transport
MSS	Maximum Segment Size
MTCP	Multipath Transmission Control Protocol
MTU	Maximum Transmission Unit
NACK	Negative Acknowledgement
NAT	Network Address Translation
NDA	Non-disclosure Agreement
NDN	Named Data Networking
NDT	Network Diagnostic Tool
NetFPGA	Network Field Programmable Gate Array
NFV	Network Functions Virtualization
NG-SDN	Next Generation Software Defined Networking
NIC	Network Interface Controller
gNMI	gRPC Network Management Interface
NN	Neural Networks
gNOI	gRPC Network Operations Interface
NOS	Network Operating System
NSF	National Science Foundation
NSH	Network Service Header
NVGRE	Network Virtualization using Generic Routing Encapsulation
OAC	Office of Advanced Cyberinfrastructure
OCP	Open Compute Project
OEM	Original Equipment Manufacturer
ONL	Open Network Linux
ONOS	Open Network Operating System
OS	Operating System
OSPF	Open Shortest Path First
OUM	Ordered Unreliable Multicast

Abbreviation	Term
OVS	Open Virtual Switch
P2P	Peer-to-peer
PBRE	Packet Buffer and Replication Engine
PBT	Postcard-Based Telemetry
PCC	Performance-oriented Congestion Control
PCC	Per-connection Consistency
PCI	Peripheral Component Interconnect
PD	Program Dependent
PGW	Packet Data Network Gateway
PI	Protocol Independent
PIE	Proportional Integral Controller Enhanced
PISA	Protocol Independent Switch Architecture
PNA	Portable NIC Architecture
PRE	Packet Replication Engine
PSA	Portable Switch Architecture
PTF	Packet Test Framework
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RDMA	Remote Direct Memory Access
RED	Random Early Detection
REST	Representational State Transfer
RFC	Request for Comments
RMT	Reconfigurable Match-action Tables
RPC	Remote Procedure Calls
RSA	Rivest-Shamir-Adleman
RSS	Really Simple Syndication
RTT	Round-trip Time
RWND	Receiver Window
SAD	Security Association Database
SAI	Switch Abstraction Interface
SAT	Boolean Satisfiability Problem
SDE	Software Development Environment
SDN	Software Defined Networking
SHA	Secure Hash Algorithm
SIGCOMM	Special Interest Group on Data Communication
SIP	Session Initiation Protocol
SLA	Service Level Agreements
SNMP	Simple Network Management Protocol
SPD	Security Policy Database
SRAM	Static Random Access Memory
SSH	Secure Shell
SSL	Secure Sockets Layer
STP	Spanning Tree Protocol
TCAM	Ternary Content-Addressable Memory
TCP	Transmission Control Protocol
TM	Traffic Management
ToR	The Onion Router
TPU	Tensor Processing Unit
TTL	Time-to-Live
UDP	User Datagram Protocol
UE	User Equipment
VIP	Virtual Internet Protocol
VLAN	Virtual Local Area Network
VMN	Verifying Mutable Networks
VN	Virtual Network
VoLTE	Voice over Long-term Evolution
VTP	Virtual Trunking Protocol
VXLAN	Virtual eXtensible Local Area Network
WAN	Wide Area Network
WCMP	Weighted-cost Multipath
XDP	eXpress Data Path
YANG	Yet Another Next Generation

REFERENCES

[1] N. McKeown, "How we might get humans out of the way." Open Networking Foundation (ONF) Connect 19, Sep. 2019. [Online]. Available: https://tinyurl.com/y4dnxacz.

[2] RFC Editor, "Number of RFCs published per year." [Online]. Available: https://www.rfc-editor.org/rfcs-per-year/.

[3] B. Trammell and M. Kuehlewind, "Report from the IAB workshop on stack evolution in a middlebox Internet (SEMI)," RFC7663. [Online]. Available: https://tools.ietf.org/html/rfc7663.

[4] G. Papastergiou, G. Fairhurst, D. Ros, A. Brunstrom, K.-J. Grinnemo, P. Hurtig, N. Khademi, M. Tüxen, M. Welzl, D. Damjanovic, and S. Mangiante, "De-ossifying the Internet transport layer: a survey and future perspectives," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 619–639, 2016.

[5] "VMware, Cisco stretch virtual LANs across the heavens." in The Register, Aug. 2011. [Online]. Available: https://tinyurl.com/y6mxhqzn.

[6] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): a framework for overlaying virtualized layer 2 networks over layer 3 networks," RFC7348. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7348.txt.

[7] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," *ACM SIGCOMM computer communication review*, vol. 37, no. 4, pp. 1–12, 2007.

[8] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: a comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[9] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, and G. Varghese, "P4: programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[10] Barefoot Networks, "Use cases." [Online]. Available: https://www.barefootnetworks.com/use-cases/.

[11] A. Weissberger, "Comcast: ONF Trellis software is in production together with L2/L3 white box switches." [Online]. Available: https://tinyurl.com/y69jc7sv.

[12] N. Akiyama, M. Nishiki, "P4 and Stratum use case for new edge cloud." [Online]. Available: https://tinyurl.com/yxuoo9qv.

[13] Stordis GmbH, "New STORDIS advanced programmable switches (APS) first to unlock the full potential of P4 and next generation software defined networking (NG-SDN)." [Online]. Available: https://tinyurl.com/y3kjnypl.

[14] Open Networking Foundation (ONF), "Stratum – ONF launches major new open source SDN switching platform with support from Google." [Online]. Available: https://tinyurl.com/yy3ykw7g.

[15] P4.org Community, "P4 gains broad adoption, joins Open Networking Foundation (ONF) and Linux Foundation (LF) to accelerate next phase of growth and innovation." [Online]. Available: https://p4.org/p4/p4-joins-onf-and-lf.html.

[16] Facebook engineering, "Disaggregate: networking recap." [Online]. Available: https://tinyurl.com/yxoaj7kw.

[17] Open Compute Project (OCP), "Alibaba DC network evolution with open SONiC and programmable HW." [Online]. Available: https://www.opencompute.org/files/OCP2018.alibaba.pdf.

[18] S. Heule, "Using P4 and P4Runtime for optimal L3 routing." [Online]. Available: https://tinyurl.com/y365gnqy.

[19] N. McKeown, "SDN phase 3: getting the humans out of the way. ONF Connect 19." [Online]. Available: https://tinyurl.com/tp9bxw4.

[20] Edgecore, "Wedge 100BF-32X, 100GbE data center switch," 2020. [Online]. Available: https://tinyurl.com/sy2jkqe.

[21] STORDIS, "The new advanced programmable switches are available." [Online]. Available: https://www.stordis.com/products/.

[22] Cisco, "Cisco Nexus 34180YC and 3464C programmable switches data sheet." [Online]. Available: https://tinyurl.com/y92cbdxe.

[23] Arista, "Arista 7170 series." [Online]. Available: https://www.arista.com/en/products/7170-series.

[24] Juniper Networks, "Juniper advancing disaggregation through P4Runtime integration." [Online]. Available: https://tinyurl.com/yygz547t.

[25] Interface Masters, "Tahoe 2624." [Online]. Available: https://interfacemasters.com/products/switches/10g-40g/tahoe-2624/.

[26] Barefoot Networks, "Tofino ASIC." [Online]. Available: https://www.barefootnetworks.com/products/brief-tofino/.

[27] Xilinx, "Xilinx solutions." [Online]. Available: https://www.xilinx.com/products/silicon-devices.html.

[28] Pensando, "The Pensando distributed services platform." [Online]. Available: https://pensando.io/our-platform/.

[29] Mellanox, "Empowering the next generation of secure cloud SmartNICs." [Online]. Available: https://www.mellanox.com/products/smartnic.

[30] Innovium, "Teralynx switch silicon." [Online]. Available: https://www.innovium.com/teralynx/.

[31] National Science Foundation: NSF. [Online]. Available: https://www.nsf.gov/.

[32] I. Baldin, J. Griffioen, K. Wang, I. Monga, A. Nikolich, "Mid-Scale RI-1 (M1:IP): FABRIC: adaptive programmable research infrastructure for computer science and science applications." [Online]. Available:

https://tinyurl.com/y463v9z9.
[33] FABRIC, "About FABRIC." [Online]. Available: https://fabric-testbed.net/about/.
[34] J. Mambretti, J. Chen, F. Yeh, and S. Y. Yu, "International P4 networking testbed," in *SC19 Network Research Exhibition*, 2019.
[35] 2STiC, "A national programmable infrastructure to experiment with next-generation networks." [Online]. Available: https://www.2stic.nl/national-programmable-infrastructure.html.
[36] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
[37] Association for Computing Machinery (ACM), "SIGCOMM 2020." [Online]. Available: https://conferences.sigcomm.org/sigcomm/2020/.
[38] H. Stubbe, "P4 compiler & interpreter: a survey," *Future Internet (FI) and Innovative Internet Technologies and Mobile Communication (IITM)*, vol. 47, 2017.
[39] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful SDN data planes," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1701–1725, 2017.
[40] W. L. da Costa Cordeiro, J. A. Marques, and L. P. Gaspary, "Data plane programmability beyond OpenFlow: opportunities and challenges for network and service operations and management," *Journal of Network and Systems Management*, vol. 25, no. 4, pp. 784–818, 2017.
[41] A. Satapathy, "Comprehensive study of P4 programming language and software-defined networks," 2018. [Online]. Available: https://tinyurl.com/y4d4zma9.
[42] R. Bifulco and 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)*, pp. 1–7, IEEE, 2018.
[43] E. Kaljic, A. Maric, P. Njemcevic, and M. Hadzialic, "A survey on data plane flexibility and programmability in software-defined networking," *IEEE Access*, vol. 7, pp. 47804–47840, 2019.
[44] P. G. Kannan and M. C. Chan, "On programmable networking evolution," *CSI Transactions on ICT*, vol. 8, no. 1, pp. 69–76, 2020.
[45] p4lang, "p4lang/p4c." [Online]. Available: https://github.com/p4lang/p4c/.
[46] p4lang/behavioral model, "The bmv2 simple switch target." [Online]. Available: https://tinyurl.com/vrasamm.
[47] H. Wang, R. Soulé, H. T. Dang, K. S. Lee, V. Shrivastav, N. Foster, and H. Weatherspoon, "P4FPGA: a rapid prototyping framework for P4," in *Proceedings of the Symposium on SDN Research*, pp. 122–135, 2017.
[48] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, and J. Rexford, "PISCES: A programmable, protocol-independent software switch," in *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 525–538, 2016.
[49] Barefoot Networks, "Technology." [Online]. Available: https://tinyurl.com/y3bdh9ah/.
[50] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: programming platform-independent stateful OpenFlow applications inside the switch," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 44–51, 2014.
[51] M. Moshref, A. Bhargava, A. Gupta, M. Yu, and R. Govindan, "Flow-level state transition as a new switch primitive for SDN," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 61–66, 2014.
[52] P4 Language Consortium, "P4Runtime." [Online]. Available: https://github.com/p4lang/PI/.
[53] J. Moy, "OSPF version 2," RFC2328. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2328.txt.
[54] H. Smit and T. Li, "Intermediate System to Intermediate System (IS-IS) extensions for traffic engineering (TE)," RFC3784. [Online]. Available: https://tools.ietf.org/html/rfc3784.
[55] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (bgp-4)," RFC4271. http://www.rfc-editor.org/rfc/rfc4271.txt.
[56] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.
[57] R. Amin, M. Reisslein, and N. Shah, "Hybrid SDN networks: a survey of existing approaches," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3259–3306, 2018.
[58] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 1–6, 2014.
[59] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: towards a model-driven SDN controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pp. 1–6, IEEE, 2014.
[60] OpenFlow switch specification, "Openflow switch specification version 1.5.1." [Online]. Available: https://tinyurl.com/y4j4a5eh.
[61] P. Garg and Y. Wang, "NVGRE: network virtualization using generic routing encapsulation," RFC7637. [Online]. Available: https://tools.ietf.org/html/rfc7637.
[62] N. McKeown, "Why does the Internet need a programmable forwarding plane." [Online]. Available: https://tinyurl.com/y6x7qqpm.
[63] N. McKeown, "SDN phase 3: getting the humans out of the way, slide 16." [Online]. Available: https://tinyurl.com/tp9bxw4.
[64] R. Cziva, "ESnet tutorial - P4 deep dive, slide 28." [Online]. Available: https://tinyurl.com/rruscv3.
[65] B. N. Vladimir Gurevich, "P4_16 introduction." [Online]. Available: https://tinyurl.com/rkkfopr.
[66] A. Shapiro, "P4-programming data plane use-cases." in P4 Expert Roundtable Series, April 28-29, 2020. [Online]. Available: https://tinyurl.com/y5n4k83h.
[67] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "RFC 7348: Virtual eXtensible Local Area Network (VXLAN): a framework for overlaying virtualized layer 2 networks over layer 3 networks," *Internet Engineering Task Force (IETF)*, 2014.
[68] C. Kim, P. Bhide, E. Doe, H. Holbrook, A. Ghanwani, D. Daly, M. Hira, and B. Davie, "In-band network telemetry (INT)," *technical specification*, 2016.
[69] C. Kim, "Evolution of networking, Networking Field Day 21, 2:01," 2019. [Online]. Available: https://tinyurl.com/y9fkj7qx.
[70] Fingerhut, Andy and Halstead, Robert and Lee, Jeongkeun and Tönsing, Johann , "P4 Tutorial." Hot Chips 2017, slide 63. [Online]. Available: https://tinyurl.com/rjjhubr.
[71] P4.org Community, "Implementing basic forwarding." [Online]. Available: https://tinyurl.com/y5uydtbn.
[72] C. E. Spurgeon, *Ethernet: the definitive guide*. "O'Reilly Media, Inc.", 2000.
[73] P4.org, "P4 language tutorial, slide 7." [Online]. Available: https://tinyurl.com/udonnx5.
[74] p4lang/p4 spec, "Portable Switch Architecture." [Online]. Available: https://tinyurl.com/uryj76k.
[75] A. Bas and A. Fingerhut, "P4 tutorial, slide 22." [Online]. Available: https://tinyurl.com/tb4m749.
[76] G. Brebner, "Extending the range of p4 programmability, slide 29." [Online]. Available: https://p4.org/assets/P4WE_2018/Gordon_Brebner.pdf.
[77] opencomputeproject/SAI, "SAI target." [Online]. Available: https://tinyurl.com/yx2g5czx.
[78] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: toward 100Gbps as research commodity," *IEEE micro*, vol. 34, no. 5, pp. 32–41, 2014.
[79] Digilent, "NetFPGA SUME Board." [Online]. Available: https://tinyurl.com/sdp6edj.
[80] NetFPGA/P4-NetFPGA-public, "SimpleSumeSwitch architecture (v1.2.1 and earlier)." [Online]. Available: https://tinyurl.com/ry46nz7.
[81] Stanford University, "CS344, development tools." [Online]. Available: https://tinyurl.com/yx6n8fzf.
[82] Barefoot Networks, "Barefoot Networks announces P4 studio." [Online]. Available: https://tinyurl.com/sv7bltn.
[83] Cascaval, C and Daly, D, "P4 architectures," 2017. [Online]. Available: https://p4.org/assets/p4-ws-2017-p4-architectures.pdf.
[84] The P4.org architecture working group, "P4_16 Portable Switch Architecture (PSA)," 2020. [Online]. Available: https://p4.org/p4-spec/docs/PSA.pdf.
[85] C. Cascone, "P4 and P4Runtime technical introduction and use cases for service providers, open networking summit 2018," 2018. [Online]. Available: https://tinyurl.com/snmcdum.
[86] Open Compute Project (OCP), "Switch Abstraction Interface (SAI)," 2020. [Online]. Available: https://github.com/opencomputeproject/SAI.
[87] SONiC, "Software for open networking in the cloud," 2020. [Online]. Available: https://azure.github.io/SONiC/.
[88] Open Compute Project (OCP), "Networking/SAI." [Online]. Available: https://www.opencompute.org/wiki/Networking/SAI.
[89] C. Cascone, "Performance evaluation of ONOS support for P4Runtime, ONF Sec & Perf Workshop TMA 2019," 2019. [Online]. Available: https://tinyurl.com/snmcdum.
[90] Hardesty, Linda, "Edgecore advances its white box ambitions, embeds

engineers in ONF projects," 2019. [Online]. Available: https://tinyurl.com/y6h2unr7.

[91] QCT, "QuantaMesh BMS T7064-IX4, the next-generation 100G spine switch for data center and cloud computing," 2020. [Online]. Available: https://tinyurl.com/y2fkmlyv.

[92] Mellanox, "Mellanox scale-out SN2000 ethernet switch series," 2020. [Online]. Available: https://www.mellanox.com/products/ethernet-switches/sn2000.

[93] ONL, "Open Network Linux," 2020. [Online]. Available: http://opennetlinux.org/.

[94] S. Choi, B. Burkov, A. Eckert, T. Fang, S. Kazemkhani, R. Sherwood, Y. Zhang, and H. Zeng, "FBOSS: building switch software at scale," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 342–356, 2018.

[95] Linux Foundation, "FRRouting (FRR)," 2020. [Online]. Available: https://frrouting.org/.

[96] S. Ibanez, G. Brebner, N. McKeown, and 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*, pp. 1–9, 2019.

[97] Pritsak, Marian, Kadosh, and Matty, "P4 compiler backend for TC." [Online]. Available: https://netdevconf.info/0x13/session.html?p4-compiler-backend-for-tc.

[98] P. Benácek, V. Pu, and H. Kubátová, "P4-to-VHDL: automatic generation of 100 Gbps packet parsers," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 148–155, IEEE, 2016.

[99] Netcope Technologies, "P4 to VHDL." [Online]. Available: https://www.netcope.com/en/products/p4-to-vhdl.

[100] W. Tu, F. Ruffy, and M. Budiu, "Linux network programming with P4," in *Linux Plumbers' Conference 2018*, 2018.

[101] p4lang/p4c, "eBPF backend." [Online]. Available: https://tinyurl.com/ullln15.

[102] opencomputeproject/SAI, "Programming NFP with P4 and C." [Online]. Available: https://tinyurl.com/sawlcmp.

[103] M. Budiu and C. Doss, "The architecture of the P4_16 compiler," 2017. [Online]. Available: https://p4.org/assets/p4-ws-2017-p4-compiler.pdf.

[104] P4.org Community, "P4 Compiler," 2020. [Online]. Available: https://p4.org/code/.

[105] Google, "Protocol buffers - Google's data interchange format," 2020. [Online]. Available: https://github.com/protocolbuffers/protobuf.

[106] P4 Language Consortium, "Lab 2: P4Runtime." [Online]. Available: https://tinyurl.com/yy6q5vzp.

[107] P4.org Community, "Packet Test Framework," 2020. [Online]. Available: https://github.com/p4lang/ptf.

[108] Biondi, P, "Scapy, packet crafting for Python2 and Python3," 2020. [Online]. Available: https://scapy.net/.

[109] B. O'Connor, C. Cascone, "Next-generation SDN tutorial." in Open Networking Foundation (ONF) Connect 2019, Sep. 2019. [Online]. Available: https://tinyurl.com/yypyod3d.

[110] Cisco, "gNMI Protocol." [Online]. Available: https://tinyurl.com/y5gfkaex.

[111] OpenConfig, "gNOI - gRPC network operations interface." [Online]. Available: https://github.com/openconfig/gnoi.

[112] OpenConfig, "Data models and APIs." [Online]. Available: https://www.openconfig.net/projects/models/.

[113] E. A. Brewer, "Kubernetes and the path to cloud native," in *Proceedings of the sixth ACM symposium on cloud computing*, pp. 167–167, 2015.

[114] Open Networking Foundation, "Stratum." [Online]. Available: https://www.opennetworking.org/stratum/.

[115] Open Networking Foundation, "NG-SDN tutorial, YANG, OpenConfig, and gNMI." [Online]. Available: https://tinyurl.com/y28nzt7p.

[116] Z. Liu, J. Bi, Y. Zhou, Y. Wang, and Y. Lin, "Netvision: towards network telemetry as a service," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pp. 247–248, IEEE, 2018.

[117] J. Hyun, N. Van Tu, and J. W.-K. Hong, "Towards knowledge-defined networking using in-band network telemetry," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–7, IEEE, 2018.

[118] Y. Kim, D. Suh, and S. Pack, "Selective in-band network telemetry for overhead reduction," in *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, pp. 1–3, IEEE, 2018.

[119] J. A. Marques, M. C. Luizelli, R. I. T. da Costa Filho, and L. P. Gaspary, "An optimization-based approach for efficient network monitoring using in-band network telemetry," *Journal of Internet Services and Applications*, vol. 10, no. 1, p. 12, 2019.

[120] B. Niu, J. Kong, S. Tang, Y. Li, and Z. Zhu, "Visualize your IP-over-optical network in realtime: a P4-based flexible multilayer in-band network telemetry (ML-INT) system," *IEEE Access*, vol. 7, pp. 82413–82423, 2019.

[121] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "PINT: probabilistic in-band network telemetry," 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*, pp. 662–680, 2020.

[122] N. Van Tu, J. Hyun, and J. W.-K. Hong, "Towards ONOS-based SDN monitoring using in-band network telemetry," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 76–81, IEEE, 2017.

[123] Serkant, "Prometheus INT exporter." [Online]. Available: https://github.com/serkantul/prometheus_int_exporter/.

[124] N. Van Tu, J. Hyun, G. Y. Kim, J.-H. Yoo, and J. W.-K. Hong, "IntCollector: a high-performance collector for in-band network telemetry," in *2018 14th International Conference on Network and Service Management (CNSM)*, pp. 10–18, IEEE, 2018.

[125] Barefoot Networks, "Barefoot Deep Insight - product brief." [Online]. Available: https://tinyurl.com/u2ncvry.

[126] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 29–42, 2017.

[127] B. Turkovic, F. Kuipers, N. van Adrichem, and K. Langendoen, "Fast network congestion detection and avoidance using P4," in *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies*, pp. 45–51, 2018.

[128] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, and M. Y. Alizadeh, Mohammad, "HPCC: high precision congestion control," in *Proceedings of the ACM Special Interest Group on Data Communication*, pp. 44–58, 2019.

[129] A. Feldmann, B. Chandrasekaran, S. Fathalli, and E. N. Weyulu, "P4-enabled network-assisted congestion feedback: a case for NACKs," 2019.

[130] E. F. Kfoury, J. Crichigno, E. Bou-Harb, D. Khoury, and G. Srivastava, "Enabling TCP pacing using programmable data plane switches," in *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, pp. 273–277, IEEE, 2019.

[131] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: A better NetFlow for data centers," in *13th {USENIX} Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 311–324, 2016.

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

[133] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim, "Language-directed hardware design for network performance monitoring," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 85–98, 2017.

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

[135] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: adaptive and fast network-wide measurements," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 561–575, 2018.

[136] N. Yaseen, J. Sonchack, and V. Liu, "Synchronized network snapshots," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 402–416, 2018.

[137] R. Joshi, T. Qu, M. C. Chan, B. Leong, and B. T. Loo, "Burstradar: practical real-time microburst monitoring for datacenter networks," in *Proceedings of the 9th Asia-Pacific Workshop on Systems*, pp. 1–8, 2018.

[138] M. Lee and J. Rexford, "Detecting violations of service-level agreements in programmable switches," 2018. [Online]. Available: https://p4campus.cs.princeton.edu/pubs/mackl_thesis_paper.pdf.

[139] J. Sonchack, O. Michel, A. J. Aviv, E. Keller, and J. M. Smith, "Scaling hardware accelerated network monitoring to concurrent and dynamic queries with* flow," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pp. 823–835, 2018.

[140] J. Sonchack, A. J. Aviv, E. Keller, and J. M. Smith, "Turboflow: Information rich flow record generation on commodity switches," in *Proceedings of the Thirteenth EuroSys Conference*, pp. 1–16, 2018.

[141] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and

W. Willinger, "Sonata: query-driven streaming network telemetry," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 357–371, 2018.

[142] X. Chen, S. L. Feibish, Y. Koral, J. Rexford, O. Rottenstreich, S. A. Monetti, and T.-Y. Wang, "Fine-grained queue measurement in the data plane," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pp. 15–29, 2019.

[143] Z. Liu, S. Zhou, O. Rottenstreich, V. Braverman, and J. Rexford, "Memory-efficient performance monitoring on programmable switches with lean algorithms," in *Symposium on Algorithmic Principles of Computer Systems (APoCS)*, 2020.

[144] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever, "Blink: fast connectivity recovery entirely in the data plane," in *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pp. 161–176, 2019.

[145] D. Ding, M. Savi, and D. Siracusa, "Estimating logarithmic and exponential functions to track network traffic entropy in P4," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2019.

[146] W. Wang, P. Tammana, A. Chen, and T. E. Ng, "Grasp the root causes in the data plane: diagnosing latency problems with SpiderMon," in *Proceedings of the Symposium on SDN Research*, pp. 55–61, 2020.

[147] R. Teixeira, R. Harrison, A. Gupta, and J. Rexford, "PacketScope: monitoring the packet lifecycle inside a switch," in *Proceedings of the Symposium on SDN Research*, pp. 76–82, 2020.

[148] J. Bai, M. Zhang, G. Li, C. Liu, M. Xu, and H. Hu, "FastFE: accelerating ML-based traffic analysis with programmable switches," in *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, SPIN '20, p. 1–7, Association for Computing Machinery, 2020.

[149] X. Chen, H. Kim, J. M. Aman, W. Chang, M. Lee, and J. Rexford, "Measuring TCP round-trip time in the data plane," in *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, pp. 35–41, 2020.

[150] Y. Qiu, K.-F. Hsu, J. Xing, and A. Chen, "A feasibility study on time-aware monitoring with commodity switches," in *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, pp. 22–27, 2020.

[151] Q. Huang, H. Sun, P. P. Lee, W. Bai, F. Zhu, and Y. Bao, "OmniMon: re-architecting Network telemetry with resource efficiency and full accuracy," 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*, pp. 404–421, 2020.

[152] X. Chen, S. Landau-Feibish, M. Braverman, and J. Rexford, "BeauCoup: answering many network traffic queries, one memory update at a time," 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*, pp. 226–239, 2020.

[153] R. Kundel, J. Blendin, T. Viernickel, B. Koldehofe, and R. Steinmetz, "P4-CoDel: active queue management in programmable data planes," in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 1–4, IEEE, 2018.

[154] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy, "Approximating fair queueing on reconfigurable switches," in *15th {USENIX} Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 1–16, 2018.

[155] S. Laki, P. Vörös, and F. Fejes, "Towards an AQM evaluation testbed with P4 and DPDK," in *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, pp. 148–150, 2019.

[156] C. Papagianni and K. De Schepper, "PI2 for P4: an active queue management scheme for programmable data planes," in *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies*, pp. 84–86, 2019.

[157] D. Bhat, J. Anderson, P. Ruth, M. Zink, and K. Keahey, "Application-based QoE support with P4 and OpenFlow," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 817–823, IEEE, 2019.

[158] S. S. Lee and K.-Y. Chan, "A traffic meter based on a multicolor marker for bandwidth guarantee and priority differentiation in sdn virtual networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 1046–1058, 2019.

[159] K. Tokmakov, M. Sarker, J. Domaschka, and S. Wesner, "A case for data centre traffic management on software programmable ethernet switches," in *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, pp. 1–6, IEEE, 2019.

[160] Y.-W. Chen, L.-H. Yen, W.-C. Wang, C.-A. Chuang, Y.-S. Liu, and C.-C. Tseng, "P4-Enabled bandwidth management," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 1–5, IEEE, 2019.

[161] M. Shahbaz, L. Suresh, J. Rexford, N. Feamster, O. Rottenstreich, and M. Hira, "Elmo: Source routed multicast for public clouds," in *Proceedings of the ACM Special Interest Group on Data Communication*, pp. 458–471, 2019.

[162] M. Kadosh, Y. Piasetzky, B. Gafni, L. Suresh, M. Shahbaz, S. Banerjee, "Realizing source routed multicast using Mellanox's programmable hardware switches, P4 Expert Roundtable Series, Apr. 2020." [Online]. Available: https://tinyurl.com/y8dfcsum.

[163] W. Braun, J. Hartmann, and M. Menth, "Scalable and reliable software-defined multicast with BIER and P4," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 905–906, IEEE, 2017.

[164] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research*, pp. 1–12, 2016.

[165] R. Miao, H. Zeng, C. Kim, J. Lee, and 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*, pp. 15–28, 2017.

[166] C. H. Benet, A. J. Kassler, T. Benson, and G. Pongracz, "MP-HULA: multipath transport aware load balancing using programmable data planes," in *Proceedings of the 2018 Morning Workshop on In-Network Computing*, pp. 7–13, 2018.

[167] V. Olteanu, A. Agache, A. Voinescu, and C. Raiciu, "Stateless datacenter load-balancing with beamer," in *15th {USENIX} Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 125–139, 2018.

[168] Z. Liu, Z. Bai, Z. Liu, X. Li, C. Kim, V. Braverman, X. Jin, and 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)*, pp. 143–157, 2019.

[169] K.-F. Hsu, P. Tammana, R. Beckett, A. Chen, J. Rexford, and D. Walker, "Adaptive weighted traffic splitting in programmable data planes," in *Proceedings of the Symposium on SDN Research*, pp. 103–109, 2020.

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

[171] E. Cidon, S. Choi, S. Katti, and N. McKeown, "AppSwitch: application-layer load balancing within a software switch," in *Proceedings of the First Asia-Pacific Workshop on Networking*, pp. 64–70, 2017.

[172] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya, "Incbricks: toward in-network computation with an in-network cache," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 795–809, 2017.

[173] C. Zhang, J. Bi, Y. Zhou, K. Zhang, and Z. Ma, "B-cache: a behavior-level caching framework for the programmable data plane," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 00084–00090, IEEE, 2018.

[174] J. Vestin, A. Kassler, and J. Åkerberg, "FastReact: in-network control and caching for industrial control networks using programmable data planes," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, pp. 219–226, IEEE, 2018.

[175] J. Woodruff, M. Ramanujam, and N. Zilberman, "P4DNS: in-network DNS," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 1–6, IEEE, 2019.

[176] R. Ricart-Sanchez, P. Malagon, P. Salva-Garcia, E. C. Perez, Q. Wang, and 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*, vol. 124, pp. 80–93, 2018.

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

[178] P. Palagummi and 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)*, pp. 338–342, IEEE, 2018.

[179] S. K. Singh, C. E. Rothenberg, G. Patra, and 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*, pp. 9–14, 2019.

[180] E. Kfoury, J. Crichigno, and E. Bou-Harb, "Offloading media traffic to programmable data plane switches," in *ICC 2020 IEEE International Conference on Communications (ICC)*, IEEE, 2020.

[181] R. Shah, V. Kumar, M. Vutukuru, and P. Kulkarni, "TurboEPC: leveraging dataplane programmability to acccelerate the mobile packet core," in *Proceedings of the Symposium on SDN Research*, pp. 83–95, 2020.

[182] T. Jepsen, M. Moshref, A. Carzaniga, N. Foster, and R. Soulé, "Packet subscriptions for programmable ASICs," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, pp. 176–183, 2018.

[183] C. Wernecke, H. Parzyjegla, G. Mühl, P. Danielis, and D. Timmermann, "Realizing content-based publish/subscribe with P4," in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 1–7, IEEE, 2018.

[184] C. Wernecke, H. Parzyjegla, G. Mühl, E. Schweissguth, and D. Timmermann, "Flexible notification forwarding for content-based publish/subscribe using P4," in *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 1–5, IEEE, 2019.

[185] R. Kundel, C. Gärtner, M. Luthra, S. Bhowmik, and B. Koldehofe, "Flexible content-based publish/subscribe over programmable data planes," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–5, IEEE, 2020.

[186] J. Li, E. Michael, N. K. Sharma, A. Szekeres, and 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)*, pp. 467–483, 2016.

[187] H. T. Dang, M. Canini, F. Pedone, and R. Soulé, "Paxos made switchy," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 2, pp. 18–24, 2016.

[188] J. Li, E. Michael, and D. R. Ports, "Eris: coordination-free consistent transactions using in-network concurrency control," in *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 104–120, 2017.

[189] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, H. Weatherspoon, M. Canini, N. Zilberman, F. Pedone, and R. Soulé, "P4xos: Consensus as a network service," tech. rep., Research Report 2018-01. USI. http://www. inf. usi. ch/research_publication. htm, 2018.

[190] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica, "Netchain: scale-free sub-rtt coordination," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pp. 35–49, 2018.

[191] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, N. Zilberman, H. Weatherspoon, M. Canini, F. Pedone, and R. Soulé, "Partitioned Paxos via the network data plane," *arXiv preprint arXiv:1901.08806*, 2019.

[192] E. Sakic, N. Deric, E. Goshi, and W. Kellerer, "P4BFT: hardware-accelerated byzantine-resilient network control plane," *arXiv preprint arXiv:1905.04064*, 2019.

[193] B. Han, V. Gopalakrishnan, M. Platania, Z.-L. Zhang, and Y. Zhang, "Network-assisted raft consensus protocol," Feb. 13 2020. US Patent App. 16/101,751.

[194] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computation is a dumb idea whose time has come," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pp. 150–156, 2017.

[195] G. Siracusano and R. Bifulco, "In-network neural networks," *arXiv preprint arXiv:1801.05731*, 2018.

[196] D. Sanvito, G. Siracusano, and R. Bifulco, "Can the network be the AI accelerator?," in *Proceedings of the 2018 Morning Workshop on In-Network Computing*, pp. 20–25, 2018.

[197] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. Ports, and P. Richtárik, "Scaling distributed machine learning with in-network aggregation," *arXiv preprint arXiv:1903.06701*, 2019.

[198] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, pp. 25–33, 2019.

[199] T. Jepsen, M. Moshref, A. Carzaniga, N. Foster, and R. Soulé, "Life in the fast lane: a line-rate linear road," in *Proceedings of the Symposium on SDN Research*, pp. 1–7, 2018.

[200] T. Kohler, R. Mayer, F. Dürr, M. Maaß, S. Bhowmik, and K. Rothermel, "P4CEP: towards in-network complex event processing," in *Proceedings of the 2018 Morning Workshop on In-Network Computing*, pp. 33–38, 2018.

[201] L. Chen, G. Chen, J. Lingys, and K. Chen, "Programmable switch as a parallel computing device," *arXiv preprint arXiv:1803.01491*, 2018.

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

[203] Y. Qiao, X. Kong, M. Zhang, Y. Zhou, M. Xu, and J. Bi, "Towards in-network acceleration of erasure coding," in *Proceedings of the Symposium on SDN Research*, pp. 41–47, 2020.

[204] Z. Yu, Y. Zhang, V. Braverman, M. Chowdhury, and X. Jin, "NetLock: fast, centralized lock management using programmable switches," 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*, pp. 126–138, 2020.

[205] S.-Y. Wang, C.-M. Wu, Y.-B. Lin, and C.-C. Huang, "High-speed data-plane packet aggregation and disaggregation by P4 switches," *Journal of Network and Computer Applications*, vol. 142, pp. 98–110, 2019.

[206] S.-Y. Wang, J.-Y. Li, and Y.-B. Lin, "Aggregating and disaggregating packets with various sizes of payload in P4 switches at 100 Gbps line rate," *Journal of Network and Computer Applications*, p. 102676, 2020.

[207] Y.-B. Lin, S.-Y. Wang, C.-C. Huang, and C.-M. Wu, "The SDN approach for the aggregation/disaggregation of sensor data," *Sensors*, vol. 18, no. 7, p. 2025, 2018.

[208] A. L. R. Madureira, F. R. C. Araújo, and L. N. Sampaio, "On supporting IoT data aggregation through programmable data planes," *Computer Networks*, p. 107330, 2020.

[209] M. Uddin, S. Mukherjee, H. Chang, and T. Lakshman, "SDN-based service automation for IoT," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pp. 1–10, IEEE, 2017.

[210] M. Uddin, S. Mukherjee, H. Chang, and T. Lakshman, "SDN-based multi-protocol edge switching for IoT service automation," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2775–2786, 2018.

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

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

[213] J. Kučera, D. A. Popescu, G. Antichi, J. Kořenek, and A. W. Moore, "Seek and push: detecting large traffic aggregates in the dataplane," *arXiv preprint arXiv:1805.05993*, 2018.

[214] D. Ding, M. Savi, G. Antichi, and D. Siracusa, "An incrementally-deployable P4-enabled architecture for network-wide heavy-hitter detection," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 75–88, 2020.

[215] R. Datta, S. Choi, A. Chowdhary, and Y. Park, "P4Guard: designing P4 based firewall," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*, pp. 1–6, IEEE, 2018.

[216] A. Almaini, A. Al-Dubai, I. Romdhani, and 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)*, pp. 58–65, IEEE, 2019.

[217] Q. Kang, L. Xue, A. Morrison, Y. Tang, A. Chen, and X. Luo, "Programmable in-network security for context-aware BYOD policies," *arXiv preprint arXiv:1908.01405*, 2019.

[218] S. Bai, H. Kim, and J. Rexford, "Passive OS fingerprinting on commodity switches,"

[219] G. Li, M. Zhang, C. Liu, X. Kong, A. Chen, G. Gu, and H. Duan, "NetHCF: enabling line-rate and adaptive spoofed IP traffic filtering," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pp. 1–12, IEEE, 2019.

[220] J. Xing, W. Wu, and A. Chen, "Architecting programmable data plane defenses into the network with FastFlex," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, pp. 161–169, 2019.

[221] Q. Kang, J. Xing, and A. Chen, "Automated attack discovery in data plane systems," in *12th {USENIX} Workshop on Cyber Security Experimentation and Test (CSET)*, 2019.

[222] A. Febro, H. Xiao, and J. Spring, "Distributed SIP DDoS defense with P4," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–8, IEEE, 2019.

[223] Â. C. Lapolli, J. A. Marques, and 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)*, pp. 19–27, IEEE, 2019.

[224] Y. Mi and A. Wang, "ML-pushback: machine learning based pushback defense against DDoS," in *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies*,

pp. 80–81, 2019.
[225] D. Scholz, S. Gallenmüller, H. Stubbe, B. Jaber, M. Rouhi, and G. Carle, "Me love (SYN-) cookies: SYN flood mitigation in programmable data planes," *arXiv preprint arXiv:2003.03221*, 2020.
[226] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: mitigating volumetric DDoS attacks with programmable switches," in *Proceedings of NDSS*, 2020.
[227] J. Xing, Q. Kang, and A. Chen, "NetWarden: mitigating network covert channels while preserving performance," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.
[228] H. M. Moghaddam and A. Mosenia, "Anonymizing masses: practical light-weight anonymity at the network level," *arXiv preprint arXiv:1911.09642*, 2019.
[229] R. Meier, P. Tsankov, V. Lenders, L. Vanbever, and M. Vechev, "NetHide: secure and practical network topology obfuscation," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 693–709, 2018.
[230] 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.
[231] T. Datta, N. Feamster, J. Rexford, and L. Wang, "{SPINE}: surveillance protection in the network elements," in *9th {USENIX} Workshop on Free and Open Communications on the Internet (FOCI)*, 2019.
[232] D. Scholz, A. Oeldemann, F. Geyer, S. Gallenmüller, H. Stubbe, T. Wild, A. Herkersdorf, and G. Carle, "Cryptographic hashing in P4 data planes," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 1–6, IEEE, 2019.
[233] F. Hauser, M. Häberle, M. Schmidt, and M. Menth, "P4-IPsec: implementation of IPsec gateways in P4 with SDN control for host-to-site scenarios," *arXiv preprint arXiv:1907.03593*, 2019.
[234] F. Hauser, M. Schmidt, M. Häberle, and M. Menth, "P4-MACsec: dynamic topology monitoring and data layer protection with MACsec in P4-based SDN," *IEEE Access*, 2020.
[235] X. Chen, "Implementing AES encryption on programmable switches via scrambled lookup tables," in *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, SPIN '20, p. 8–14, Association for Computing Machinery, 2020.
[236] C. Zhang, J. Bi, Y. Zhou, J. Wu, B. Liu, Z. Li, A. B. Dogar, and Y. Wang, "P4DB: on-the-fly debugging of the programmable data plane," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pp. 1–10, IEEE, 2017.
[237] Y. Zhou, J. Bi, Y. Lin, Y. Wang, D. Zhang, Z. Xi, J. Cao, and C. Sun, "P4tester: efficient runtime rule fault detection for programmable data planes," in *Proceedings of the International Symposium on Quality of Service*, pp. 1–10, 2019.
[238] M. V. Dumitru, D. Dumitrescu, and C. Raiciu, "Can we exploit buggy P4 programs?," in *Proceedings of the Symposium on SDN Research*, pp. 62–68, 2020.
[239] S. Kodeswaran, M. T. Arashloo, P. Tammana, and J. Rexford, "Tracking P4 program execution in the data plane," in *Proceedings of the Symposium on SDN Research*, pp. 117–122, 2020.
[240] N. Lopes, N. Bjørner, N. McKeown, A. Rybalchenko, D. Talayco, and G. Varghese, "Automatically verifying reachability and well-formedness in P4 networks," *Technical Report, Tech. Rep*, 2016.
[241] L. Freire, M. Neves, L. Leal, K. Levchenko, A. Schaeffer-Filho, and M. Barcellos, "Uncovering bugs in P4 programs with assertion-based verification," in *Proceedings of the Symposium on SDN Research*, pp. 1–7, 2018.
[242] M. Neves, L. Freire, A. Schaeffer-Filho, and M. Barcellos, "Verification of P4 programs in feasible time using assertions," in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, pp. 73–85, 2018.
[243] J. Liu, W. Hallahan, C. Schlesinger, M. Sharif, J. Lee, R. Soulé, H. Wang, C. Caşcaval, N. McKeown, and N. Foster, "P4v: practical verification for programmable data planes," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 490–503, 2018.
[244] A. Nötzli, J. Khan, A. Fingerhut, C. Barrett, and P. Athanas, "P4pktgen: automated test case generation for P4 programs," in *Proceedings of the Symposium on SDN Research*, pp. 1–7, 2018.
[245] D. Lukács, M. Tejfel, and G. Pongrácz, "Keeping P4 switches fast and fault-free through automatic verification," *Acta Cybernetica*, vol. 24, no. 1, pp. 61–81, 2019.
[246] R. Stoenescu, D. Dumitrescu, M. Popovici, L. Negreanu, and C. Raiciu, "Debugging P4 programs with Vera," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 518–532, 2018.
[247] A. Shukla, K. N. Hudemann, A. Hecker, and S. Schmid, "Runtime verification of P4 switches with reinforcement learning," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, pp. 1–7, 2019.
[248] D. Dumitrescu, R. Stoenescu, L. Negreanu, and 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*, pp. 571–585, 2020.
[249] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, 2015.
[250] C. Hopps *et al.*, "Analysis of an equal-cost multi-path algorithm," tech. rep., RFC 2992, November, 2000.
[251] S. Sinha, S. Kandula, and D. Katabi, "Harnessing TCP's burstiness with flowlet switching," in *Proc. 3rd ACM Workshop on Hot Topics in Networks (Hotnets-III)*, Citeseer, 2004.
[252] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2015.
[253] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, "Fast packet processing with eBPF and XDP: concepts, code, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, 2020.
[254] Broadcom, "BroadView Analytics, Trident 3 in-band telemetry." [Online]. Available: https://tinyurl.com/yxr2qydb.
[255] J. Crichigno, E. Bou-Harb, and N. Ghani, "A comprehensive tutorial on science DMZ," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 2041–2078, 2018.
[256] J. F. Kurose and K. W. Ross, "Computer networking a top down approach featuring the intel," 2016.
[257] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.
[258] D. Leith and R. Shorten, "H-TCP: TCP congestion control for high bandwidth-delay product paths," *draft-leith-tcp-htcp-06 (work in progress)*, 2008.
[259] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: congestion-based congestion control," *Communications of the ACM*, vol. 60, no. 2, pp. 58–66, 2017.
[260] S. Floyd, "TCP and explicit congestion notification," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 5, pp. 8–23, 1994.
[261] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based congestion control for the data center," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 537–550, 2015.
[262] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale RDMA deployments," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 523–536, 2015.
[263] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 conference*, pp. 63–74, 2010.
[264] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 435–446, 2013.
[265] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "{PCC}: Re-architecting congestion control for consistent high performance," in *12th {USENIX} Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 395–408, 2015.
[266] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, *et al.*, "The QUIC transport protocol: design and Internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 183–196, 2017.
[267] P. Cheng, F. Ren, R. Shu, and C. Lin, "Catch the whole lot in an action: rapid precise packet loss notification in data center," in *11th {USENIX} Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 17–28, 2014.
[268] A. Ramachandran, S. Seetharaman, N. Feamster, and V. Vazirani, "Fast monitoring of traffic subpopulations," in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pp. 257–270, 2008.
[269] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," *Journal of Computer and system sciences*, vol. 58, no. 1, pp. 137–147, 1999.

[270] V. Braverman and R. Ostrovsky, "Zero-one frequency laws," in *Proceedings of the forty-second ACM symposium on Theory of computing*, pp. 281–290, 2010.

[271] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *International Colloquium on Automata, Languages, and Programming*, pp. 693–703, Springer, 2002.

[272] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[273] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM journal on computing*, vol. 31, no. 6, pp. 1794–1813, 2002.

[274] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on networking*, vol. 1, no. 4, pp. 397–413, 1993.

[275] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pp. 29–42, 2013.

[276] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.

[277] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pp. 139–152, 2015.

[278] C. Dovrolis, K. P. Gummadi, A. Kuzmanovic, and S. D. Meinrath, "Measurement lab: overview and an invitation to the research community," *ACM Sigcomm Computer Communication Review*, vol. 40, no. 3, pp. 53–56, 2010.

[279] B. Tierney, J. Metzger, J. Boote, E. Boyd, A. Brown, R. Carlson, M. Zekauskas, J. Zurawski, M. Swany, and M. Grigoriev, "perfSONAR: instantiating a global network measurement framework," *SOSP Wksp. Real Overlays and Distrib. Sys*, 2009.

[280] R. Cziva, C. Lorier, and D. P. Pezaros, "Ruru: high-speed, flow-level latency measurement and visualization of live Internet traffic," in *Proceedings of the SIGCOMM Posters and Demos*, pp. 46–47, 2017.

[281] R. Dolby, "Noise reduction systems," Nov. 5 1974. US Patent 3,846,719.

[282] S. V. Vaseghi, *Advanced digital signal processing and noise reduction*. John Wiley & Sons, 2008.

[283] P. Flajolet, D. Gardy, and L. Thimonier, "Birthday paradox, coupon collectors, caching algorithms and self-organizing search," *Discrete Applied Mathematics*, vol. 39, no. 3, pp. 207–229, 1992.

[284] J. Gettys, "Bufferbloat: dark buffers in the Internet," *IEEE Internet Computing*, no. 3, p. 96, 2011.

[285] M. Allman, "Comments on bufferbloat," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 1, pp. 30–37, 2013.

[286] Y. Gong, D. Rossi, C. Testa, S. Valenti, and M. D. Täht, "Fighting the bufferbloat: on the coexistence of AQM and low priority congestion control," *Computer Networks*, vol. 65, pp. 255–267, 2014.

[287] C. Staff, "Bufferbloat: what's wrong with the Internet?," *Communications of the ACM*, vol. 55, no. 2, pp. 40–47, 2012.

[288] V. G. Cerf, "Bufferbloat and other internet challenges," *IEEE Internet Computing*, vol. 18, no. 5, pp. 80–80, 2014.

[289] F. Schwarzkopf, S. Veith, and M. Menth, "Performance analysis of CoDel and PIE for saturated TCP sources," in *2016 28th International Teletraffic Congress (ITC 28)*, vol. 1, pp. 175–183, IEEE, 2016.

[290] A. Mushtaq, R. Mittal, J. McCauley, M. Alizadeh, S. Ratnasamy, and S. Shenker, "Datacenter congestion control: identifying what is essential and making it practical," *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 3, pp. 32–38, 2019.

[291] K. De Schepper, O. Bondarenko, I.-J. Tsang, and B. Briscoe, "Pi2: a linearized AQM for both classic and scalable TCP," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pp. 105–119, 2016.

[292] K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers," RFC8376. [Online]. Available: https://tools.ietf.org/html/rfc8376.

[293] B. Fenner, M. Handley, H. Holbrook, I. Kouvelas, R. Parekh, Z. Zhang, and L. Zheng, "Protocol independent multicast-sparse mode (PIM-SM): protocol specification (revised).," [Online]. Available: https://tools.ietf.org/html/rfc7761.

[294] H. Holbrook, B. Cain, and B. Haberman, "Using Internet group management protocol version 3 (IGMPv3) and multicast listener discovery protocol version 2 (MLDv2) for source-specific multicast," *RFC 4604 (Proposed Standard), Internet Engineering Task Force*, 2006.

[295] I. Wijnands, E. C. Rosen, A. Dolganow, T. Przygienda, and S. Aldrin, "Multicast using bit index explicit replication (BIER)," in *RFC Editor*, 2017.

[296] B. Carpenter and S. Brim, "Middleboxes: taxonomy and issues," 2002. [Online]. Available: https://tools.ietf.org/html/rfc3234.

[297] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, *et al.*, "CONGA: distributed congestion-aware load balancing for datacenters," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, pp. 503–514, 2014.

[298] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, *et al.*, "TCP extensions for multipath operation with multiple addresses." [Online]. Available: https://tools.ietf.org/html/rfc8376.

[299] J. McCauley, A. Panda, A. Krishnamurthy, and S. Shenker, "Thoughts on load distribution and the role of programmable switches," *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 1, pp. 18–23, 2019.

[300] S. Signorello, R. State, J. François, and O. Festor, "NDN.p4: programming information-centric data-planes," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pp. 384–389, IEEE, 2016.

[301] G. Grigoryan and Y. Liu, "PFCA: a programmable FIB caching architecture," in *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems*, pp. 97–103, 2018.

[302] T. Norp, "5G Requirements and key performance indicators," *Journal of ICT Standardization*, vol. 6, no. 1, pp. 15–30, 2018.

[303] A. Mahmood, C. Casetti, C. F. Chiasserini, P. Giaccone, and J. Härri, "Efficient caching through stateful SDN in named data networking," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 1, p. e3271, 2018.

[304] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A survey of information-centric networking research," *IEEE communications surveys & tutorials*, vol. 16, no. 2, pp. 1024–1049, 2013.

[305] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," in *Proceedings DARPA Active Networks Conference and Exposition*, pp. 2–15, IEEE, 2002.

[306] E. F. Kfoury, J. Gomez, J. Crichigno, E. Bou-Harb, and D. Khoury, "Decentralized distribution of PCP mappings over blockchain for end-to-end secure direct communications," *IEEE Access*, vol. 7, pp. 110159–110173, 2019.

[307] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*, pp. 307–320, 2006.

[308] L. Lamport *et al.*, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.

[309] Marvell, "OCTEON II CN68XX family of multi-core MIPS64 processors." [Online]. Available: https://tinyurl.com/vcr4lal.

[310] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 {USENIX} Annual Technical Conference (USENIX ATC 14)*, pp. 305–319, 2014.

[311] Huynh Tu Dang, "Consensus as a network service." [Online]. Available: https://tinyurl.com/y2t9plsu.

[312] J. Nelson, "SwitchML scaling distributed machine learning with in-network aggregation." [Online]. Available: https://tinyurl.com/y53upm7k.

[313] F. Yang, Z. Wang, X. Ma, G. Yuan, and X. An, "SwitchAgg: a further step towards in-network computation," *arXiv preprint arXiv:1904.04024*, 2019.

[314] D. Das, S. Avancha, D. Mudigere, K. Vaidynathan, S. Sridharan, D. Kalamkar, B. Kaul, and P. Dubey, "Distributed deep learning using synchronous stochastic gradient descent," *arXiv preprint arXiv:1602.06709*, 2016.

[315] S. Farrell, "Low-power wide area network (LPWAN) overview," RFC8376. [Online]. Available: https://tools.ietf.org/html/rfc8376.

[316] A. Koike, T. Ohba, and R. Ishibashi, "IoT network architecture using packet aggregation and disaggregation," in *2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, pp. 1140–1145, IEEE, 2016.

[317] J. Deng and M. Davis, "An adaptive packet aggregation algorithm for wireless networks," in *2013 International Conference on Wireless Communications and Signal Processing*, pp. 1–6, IEEE, 2013.

[318] Y. Yasuda, R. Nakamura, and H. Ohsaki, "A probabilistic interest packet aggregation for content-centric networking," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 783–788, IEEE, 2018.

[319] A. S. Akyurek and T. S. Rosing, "Optimal packet aggregation schedul-

ing in wireless networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 12, pp. 2835–2852, 2018.

[320] K. Zhou and N. Nikaein, "Packet aggregation for machine type communications in LTE with random access channel," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 262–267, IEEE, 2013.

[321] A. Majeed and N. B. Abu-Ghazaleh, "Packet aggregation in multi-rate wireless LANs," in *2012 9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pp. 452–460, IEEE, 2012.

[322] D. SIG, "Bluetooth core specification version 4.2," *Specification of the Bluetooth System*, 2014.

[323] S. Farahani, *ZigBee wireless networks and transceivers*. Newnes, 2011.

[324] O. Hersent, D. Boswarthick, and O. Elloumi, *The Internet of things: key applications and protocols*. John Wiley & Sons, 2011.

[325] J. Shi, W. Quan, D. Gao, M. Liu, G. Liu, C. Yu, and W. Su, "Flowlet-based stateful multipath forwarding in heterogeneous Internet of things," *IEEE Access*, vol. 8, pp. 74875–74886, 2020.

[326] S. Do, L.-V. Le, B.-S. P. Lin, and L.-P. Tung, "SDN/NFV-based network infrastructure for enhancing IoT gateways," in *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1135–1142, IEEE, 2019.

[327] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *International Conference on Database Theory*, pp. 398–412, Springer, 2005.

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

[329] S. Heule, M. Nunkesser, and A. Hall, "HyperLogLog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proceedings of the 16th International Conference on Extending Database Technology*, pp. 683–692, 2013.

[330] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Anonymous connections and onion routing," *IEEE Journal on Selected areas in Communications*, vol. 16, no. 4, pp. 482–494, 1998.

[331] V. Liu, S. Han, A. Krishnamurthy, and T. Anderson, "Tor instead of IP," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, pp. 1–6, 2011.

[332] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig, "HORNET: high-speed onion routing at the network layer," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1441–1454, 2015.

[333] M. Zalewski and W. Stearns, "p0f," *see http://lcamtuf. coredump. cx/p0f3*, 2006.

[334] J. Barnes and P. Crowley, "k-p0f: A high-throughput kernel passive OS fingerprinter," in *Architectures for Networking and Communications Systems*, pp. 113–114, IEEE, 2013.

[335] S. Hong, R. Baykov, L. Xu, S. Nadimpalli, and G. Gu, "Towards SDN-defined programmable BYOD (bring your own device) security," in *NDSS*, 2016.

[336] S. Hilton, "Dyn analysis summary of Friday October 21 Attack, 2016.." [Online]. Available: https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/.

[337] S. Kottler, "February 28th DDoS incident report, March, 2018." [Online]. Available: https://githubengineering.com/ddos-incident-report/.

[338] K. Friday, E. Kfoury, E. Bou-Harb, and J. Crichigno, "Towards a unified in-network DDoS detection and mitigation strategy," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pp. 218–226, 2020.

[339] J. Ioannidis and S. M. Bellovin, "Implementing pushback: router-based defense against DDoS attacks," in *NDSS*, 2016.

[340] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, "I know what your packet did last hop: using packet histories to troubleshoot networks," in *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, pp. 71–85, 2014.

[341] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao, and H. Zheng, "Packet-level telemetry in large datacenter networks," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pp. 479–491, 2015.

[342] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, "Automatic test packet generation," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pp. 241–252, 2012.

[343] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: static checking for networks," in *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pp. 113–126, 2012.

[344] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: verifying network-wide invariants in real time," in *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 15–27, 2013.

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

[346] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King, "Debugging the data plane with Anteater," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 290–301, 2011.

[347] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 99–111, 2013.

[348] A. Horn, A. Kheradmand, and M. Prasad, "Delta-net: real-time network verification using atoms," in *14th {USENIX} Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 735–749, 2017.

[349] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Model checking invariant security properties in OpenFlow," in *2013 IEEE international conference on communications (ICC)*, pp. 1974–1979, IEEE, 2013.

[350] A. Panda, O. Lahav, K. Argyraki, M. Sagiv, and S. Shenker, "Verifying reachability in networks with mutable datapaths," in *14th {USENIX} Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 699–718, 2017.

[351] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, *et al.*, "The design and implementation of open vswitch," in *12th {USENIX} Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 117–130, 2015.

[352] Barefoot Networks, "Barefoot Academy," 2020. [Online]. Available: https://www.barefootnetworks.com/barefoot-academy/.

[353] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 99–110, 2013.

[354] S. Chole, A. Fingerhut, S. Ma, A. Sivaraman, S. Vargaftik, A. Berger, G. Mendelson, M. Alizadeh, S.-T. Chuang, I. Keslassy, *et al.*, "dRMT: disaggregated programmable switching," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 1–14, 2017.

[355] R. Pagh and F. F. Rodler, "Cuckoo hashing," *J. Algorithms*, vol. 51, p. 122–144, May 2004.

[356] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[357] D. Kim, Y. Zhu, C. Kim, J. Lee, and S. Seshan, "Generic external memory for switch data planes," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, pp. 1–7, 2018.

[358] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan, "TEA: enabling state-intensive network functions on programmable switches," in *Proceedings of the 2020 ACM SIGCOMM Conference*, 2020.

[359] Y. Tokusashi, H. T. Dang, F. Pedone, R. Soulé, and N. Zilberman, "The case for in-network computing on demand," in *Proceedings of the Fourteenth EuroSys Conference 2019*, pp. 1–16, 2019.

[360] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker, "SNAP: stateful network-wide abstractions for packet processing," in *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 29–43, 2016.

[361] G. Sviridov, M. Bonola, A. Tulumello, P. Giaccone, A. Bianco, and G. Bianchi, "LODGE: Local decisions on global states in programmable data planes," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pp. 257–261, IEEE, 2018.

[362] G. Sviridov, M. Bonola, A. Tulumello, P. Giaccone, A. Bianco, and G. Bianchi, "Local decisions on replicated states (LOADER) in programmable data planes: programming abstraction and experimental evaluation," *arXiv preprint arXiv:2001.07670*, 2020.

[363] S. Luo, H. Yu, and L. Vanbever, "Swing state: consistent updates for stateful and programmable data planes," in *Proceedings of the Symposium on SDN Research*, pp. 115–121, 2017.

[364] J. Xing, A. Chen, and T. E. Ng, "Secure state migration in the data

plane," in *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, pp. 28–34, 2020.

[365] E. F. Kfoury, J. Gomez, J. Crichigno, and E. Bou-Harb, "An emulation-based evaluation of TCP BBRv2 alpha for wired broadband," *Computer Communications*, 2020.