Computer Communications 170 (2021) 109–129

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A review on P4-Programmable data planes: Architecture, research efforts, and future directions

Sukhveer Kaur *, Krishan Kumar, Naveen Aggarwal

University Institute of Engineering and Technology (UIET), Panjab University, Chandigarh, India

ARTICLE INFO

Keywords:
P4
Programmable data plane
Software defined networking
OpenFlow
Control plane
Data plane
DDoS
Monitoring
Load balancing

ABSTRACT

Software Defined Networking (SDN) is a promising technology that provides flexibility, programmability, and network automation by shifting the network intelligence to the logically centralized controller. In SDN architecture, the controller maintains the global view of network topology; therefore the network management is efficient as compared to the traditional networks. Moreover, the cost of SDN devices is less due to the use of open source network operating system instead of proprietary and vendor-specific software. In spite of the flexibility offered by the SDN, OpenFlow enabled switches have a fixed behavior as specified in the datasheet of switch ASIC. They recognize fixed set of header fields according to the support of OpenFlow version. In addition, SDN is suffering from scalability and performance issues because SDN switches heavily dependent on the control plane to forward the packets which increases the data-control communication overhead. To resolve these issues, P4-Programmable data plane switches can be used. The analysis of review articles about SDN suggests insufficient focus on the data plane programmability. Therefore, this paper provides the comprehensive overview of domain-specific language (P4) for the programmability of the data plane. We have discussed the problems in the traditional SDN architecture and then, how these problems can be managed by the data plane programmability. Further, this study critically analyze the research articles based on the P4 programming language for network traffic monitoring, DDoS attack detection, load balancing, and packet aggregation and disaggregation. Finally, we have identified the research gaps and highlighted the open research challenges in the field of data plane programmability for the future directions.

1. Introduction

Emerging technologies in the information and communication technology (ICT) domain, in particular, 5G, Cloud Computing, Big Data, Network Function Virtualization (NFV), Internet of Things (IoT), and Intent based networking explicit the need of high bandwidth, ubiquitous accessibility, and dynamic management of computer networks. However, the traditional network architectures are not able to fulfill these demands due to the hard coupling between network operating system and data plane. In addition, management and maintenance of the traditional network devices are difficult and time consuming as the interfaces are varied across vendors. Moreover, these devices are suffered from the network ossification problem which has led to the major barrier in IPv6 deployment. So, it led to increase in the network management and maintenance cost. These problems can be resolved with the SDN technology that decouples the network operating system (control plane) from the forwarding hardware (data plane) [1]. OpenFlow [2] is the standard protocol that is used for the communication between the data and control plane.

SDN has emerged as a promising architecture that provides the centralized management of the network devices. It makes the network dynamic, flexible, and programmable by shifting the network control logic to the logically centralized controller [3]. In SDN, the main role of data plane switches is to forward the packets according to the instructions given by the controller that makes the management of the network easier as compared to the traditional architecture. Just like high level language that abstract the complexity of machine language, SDN abstract the complexity of individual management of each network device. Moreover, this framework provides several benefits, which include (1) Network automation, (2) Reduced CAPEX and OPEX cost, (3) Increase reliability and security of the network, (4) Centralized network management, (5) Standard protocol for configuring the devices from multiple vendors. SDN is gaining a significant attraction from both the industry and academic world. The companies like Google, AT&T, Microsoft, and Cisco has moved towards the SDN to reduce the network cost and to speed up the delivery of client-oriented services.

In spite of the immense advantages of the SDN architecture, it is suffering from scalability and performance issues due to the lack of intelligence in the SDN switches (OpenFlow enabled switches) for stateful packet processing. The switches in the data plane heavily dependent on the controller for network traffic forwarding/monitoring

* Corresponding author.
E-mail addresses: bhullarsukh96@gmail.com (S. Kaur), k.salujauiet@gmail.com (K. Kumar), navagg@gmail.com (N. Aggarwal).

https://doi.org/10.1016/j.comcom.2021.01.027
Received 5 September 2020; Received in revised form 23 November 2020; Accepted 21 January 2021
Available online 29 January 2021

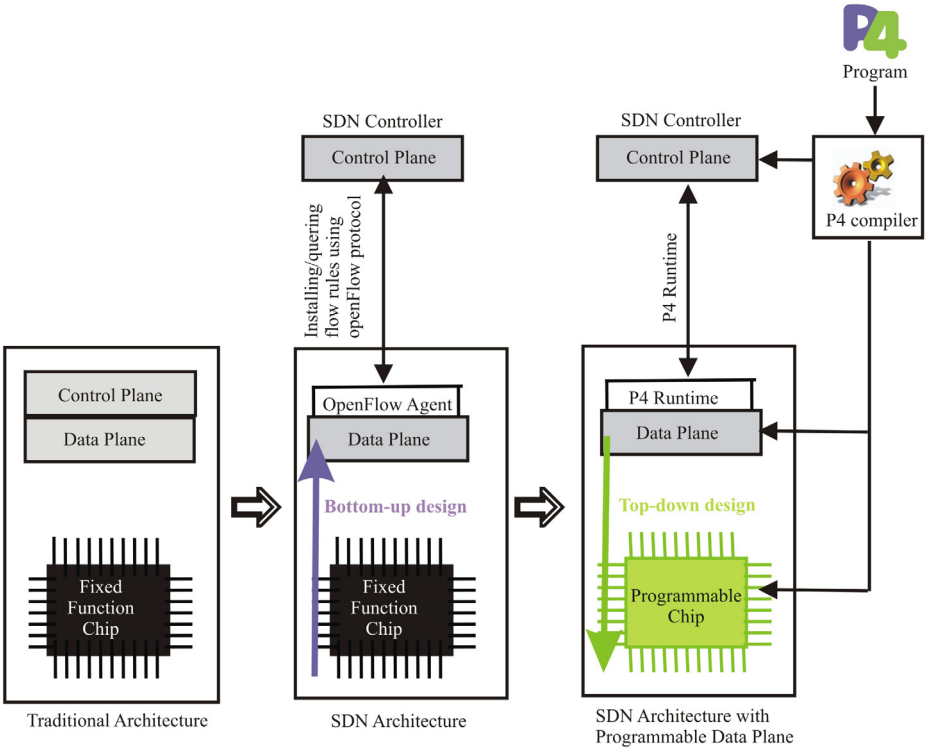

Fig. 1. Evolution from the traditional network architecture to the combined SDN model with programmable data planes [7].

Table 1
No. of matching fields in different OF versions [8].

OF version	Release date	Match fields
1.0	Dec, 2009	12
1.1	Feb, 2011	15
1.2	Dec, 2011	36
1.3	Jun, 2012	40
1.4	Oct, 2013	41
1.5	Dec, 2014	44

that introduced a data-control plane communication overhead [4,5]. Although openflow enabled data planes provides partial support for stateful packet processing with the help of counters, but it does not maintain the state information. Moreover, OpenFlow enabled switches have a fixed behavior as specified in the datasheet of switch ASIC. They recognize fixed set of header fields according to the support of OpenFlow version and process the packets using small set of predefined actions. The network administrators cannot extend the header fields and actions to meet the requirements of various applications [6]. The OpenFlow starts with 12 matching header fields and now it supports up-to 44 header fields as shown in Table 1. The complexity of OpenFlow specification is increasing from version 1.0 to 1.5 but still not providing the flexibility to add new header fields. The proliferation of new header fields shows no signs of stopping. To solve all the above problems, the Google, Microsoft, Intel, Stanford, Princeton, and Barefoot defined the domain-specific language P4 (Programming Protocol-independent Packet Processors) for the programmability of the data plane. In the combined SDN model with programmable data planes, apart from switch's initial functionalities, additional functions and packet processing details can be programmed using P4. Using P4, we got flexibility to define different header structures and corresponding functions for matching and defining actions that switch can take on each packet. The flexibility of programmable pipeline is the key advantage of P4 switch over OpenFlow switch. The evolution from the traditional network architecture to the combined SDN model with programmable data planes is shown in Fig. 1.

There are many good survey papers that provide a good overview of SDN framework [9–15], and its various aspects such as security [16–21], scalability [22,23], QoS [24,25], hybrid SDN [26–28], and traffic engineering [29,30]. It has been observed that most of SDN research focuses on more control plane programmability and less focus on the programmability of the data plane. A few survey papers exist that focus on the programmable data planes. Cordeiro et al. (2017) [31] discussed the literature of programmable data planes from active networks to the state-of-art solutions. They also provided an overview on how the programmable data planes can improve the services like network monitoring and security. In another related study (2018) [32], authors have provided the anatomy of programmable switch and discussed various open issues in the programmable data plane technologies. Kaljic et al. (2019) [33] discussed the generic approaches to improve the data plane flexibility and programmability. The recent short survey paper (2020) [34] focused on the evolution of programmable data planes from the traditional SDN framework. However, above mentioned survey papers provided the general overview of the data plane programmability, they are not specific to P4. Since the P4 was introduced in 2014, it has been used in a wide number of research areas such as network monitoring, DDoS detection, load balancing, and packet aggregation, by putting the intelligence into the data plane. There is an urgent need to review all the research studies in the field of P4-Programmable data planes owing to its huge potential and growing demand in the telecommunication industry [35]. Moreover, there is a need to discuss the various open research challenges in this field to provide the future research directions. To the best of our knowledge, this is the first review article that provided an in-depth overview of P4 domain-specific language for data plane programmability. Table 2 shows the comparison of proposed survey with the existing studies. The main contributions of the paper are:

- Analyzed and highlighted the major factors which lead to the evolution of P4-Programmable SDN architecture from the traditional network architecture.
- Detailed working flow model of P4-Programmable data planes is presented along with the different compiling tools supported by various P4 architectures and targets.

Table 2
Comparison of proposed survey with the existing surveys.

Ref	Objective	Evolution of programmable data planes	P4 Architecture	P4 compiling tools	Categorization of research efforts with programmable data planes				Research Gaps	Research Challenges
					Network monitoring DDoS attacks detection	Load balancing	Packet aggregation			
[31] 2017	Surveyed the literature on programmable data planes	✓	✓	×	✓	×	×	×	×	✓
[32] 2018	Presented the survey on recent trends and open issues in programmable devices	✓	✓	×	×	×	×	×	×	✓
[33] 2019	Reviewed various approaches to improve data plane programmability	✓	×	×	×	×	×	×	×	✓
[34] 2020	Presented a survey on the evolution and applications of programmable network	✓	×	×	✓	×	×	×	×	×
Proposed Survey	Critical examination of research efforts with P4-Programmable data planes	✓	✓	✓	✓	✓	✓	✓	✓	✓
	✓: Full detail			× : Partial detail				× : None		

- Proposed a novel way to categorized the P4-based solutions using different parameters such as “Network monitoring”, “DDoS attack detection”, “Load balancing”, and “Packet aggregation” by highlighting the major strengths and weaknesses.
- Identified the key research challenges in the field of data plane programmability.

The outlines of paper are: In Section 2, we have discussed the underlying architecture of P4-Programmable data planes. Section 3 describes recent research efforts that used P4- Programmable data planes for efficient network monitoring, DDoS attack detection, load balancing, and packet aggregation and disaggregation. We identified the various research gaps in the existing research articles in Section 4. Section 5 highlights the open research challenges associated with data plane programmability and Section 6, conclude the paper with a discussion of future research directions.

2. P4 (programming protocol-independent packet processors)

P4 [6] is a language designed to change the packet forwarding behavior of the SDN switches. This language is used to program the hardware just like Verilog and VHDL languages, but unlike Verilog and VHDL, one does not need to understand the underlying hardware details. Initially, it was designed to program the hardware or software switches, but now it covers the wide variety of devices such as network interface cards, network appliances, ASIC, NPU and FPGA. So, the generic term “target” is used to represent all such devices. P4 has a capability to define the custom header formats and parse these headers dynamically from the packet [6]. In addition, it supports the custom match+action tables and other forwarding related constructs such as counters, registers, header field manipulations. This makes the P4 language completely protocol independent. If a new protocol is deployed in the network, it is easy to change the P4 program to support the new header fields. Another feature of the P4 is re-configurability, because the new P4 program can be deployed easily when required, instead of purchasing the new hardware. Moreover, the P4 is free from any target specific features which make the language target independence. However, P4 depends on the architecture of the device that is provided by the vendors. The P4 program written for a specific architecture should be portable across all targets that implement the same architectural model [38]. The targets may contain both control plane and data plane, but the P4 is only designed to program the data plane. P4 can also be used to partially define the interface between control and data plane, but it cannot be used to define the functionality of the control plane. Fig. 2 shows the working flow model of P4 programmable data plane.

2.1. Headers format definition

The header types of P4 are similar to the C struct. The best feature of P4 language is that, it not only allow the processing of standard packet headers (Ethernet, IP, TCP, UDP etc.) but also allow the processing of custom headers. All types of headers (both standard and custom) and the sequences they are parsed must be explicitly defined in the P4 program.

2.2. P4 program pipeline

The pipeline of P4 program contains a number of functional blocks which process the input data and pass it to the next block in a pipeline. This section explains how these function blocks are programmed by the P4 language (Fig. 3).

2.2.1. Parser

The job of the parser is to specify how the headers of the packet will be parsed. It correctly identifies the headers present in the incoming packet. The parser is represented by a finite state machine where the states are responsible for extracting the individual header struct into runtime variables. Transitions can be made conditionally based on the value of a parsed header field. Fig. 4 shows the parsed representation of Ethernet header in which parser checks the ethertype field and continues parsing IPv4 header if ethertype matches with IPv4 value. After parsing all the headers in the packet according to the specification, it sends the parsed packet to the first control block for further processing.

2.2.2. Ingress and egress control blocks

It contains multiple match+action tables that match the packet based on different header fields and decide what actions to be performed when a match occur (Fig. 4). It also defines a control function that decides the order in which the tables should be executed. The ingress pipeline performs the packet modification and decides the output port to forward the packet. The egress pipeline performs further necessary header modification such as rewriting the source MAC address with the egress port MAC address of the switch.

2.2.3. Deparser

The job of deparser is to serialize the modified headers back into the packet and send it to the next switch. This function takes two arguments: packet_out variable and header values modified by the previous control blocks in the pipeline as shown in Fig. 4. The emit function of packet_out specifies the order in which the headers should be serialized. This order can be independent of the order in which the packet headers were parsed by the parser block.

2.3. P4 compilers

The target manufactures are responsible for providing the architecture definition and P4 compiler for the specific target machine. The job of P4 compiler is to translate the P4 code into a target-specific representation. P4 compiler consists of two parts: front end that convert the P4 code into target independent representation (IR) and back-end that map the IR to the specific target. Table 3 shows the compiling tools for various P4 targets and architectural model. The P4 compiler produces two outputs (1) A configuration that implements the forwarding behavior in the switches according to the P4 input program (2) An API that is used by the controller to control the functionality of data plane switches.

2.4. P4 architecture

The target-specific P4 architectural model identifies the functional blocks that are present in the data plane pipeline. It also defines the interfaces for each block and their capabilities. In general, a data plane pipeline may contain both fixed-function and programmable blocks. The P4 programmer is only responsible to define the behavior of programmable blocks because fixed function blocks are defined by the target manufactures. It is to be noted that P4 programs are not portable across different architectures. However, programs are expected to be portable across all targets that define the same architecture model. For instance, SimpleSumeSwitch architecture is used by NetFPGA based devices [39], SWArch architecture is used by software switches like BMv2 [40], Programmable switch ASICs like Barefoot Tofino [41] implements the ASICArch architecture (Fig. 5).

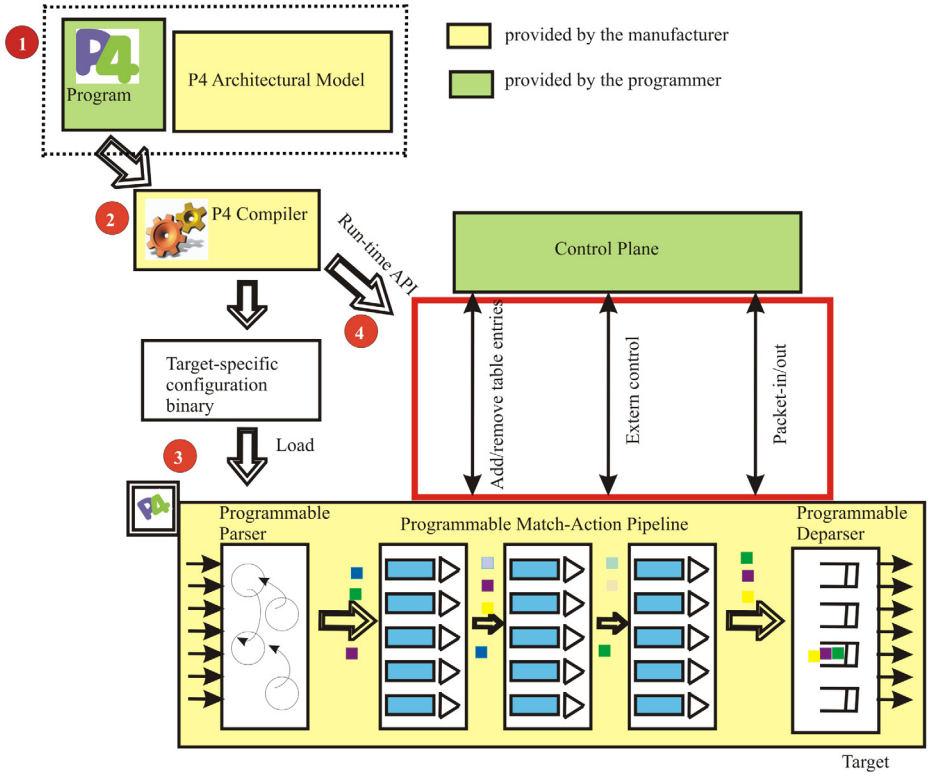

Fig. 2. Working flow model of P4-Programmable data plane [36].

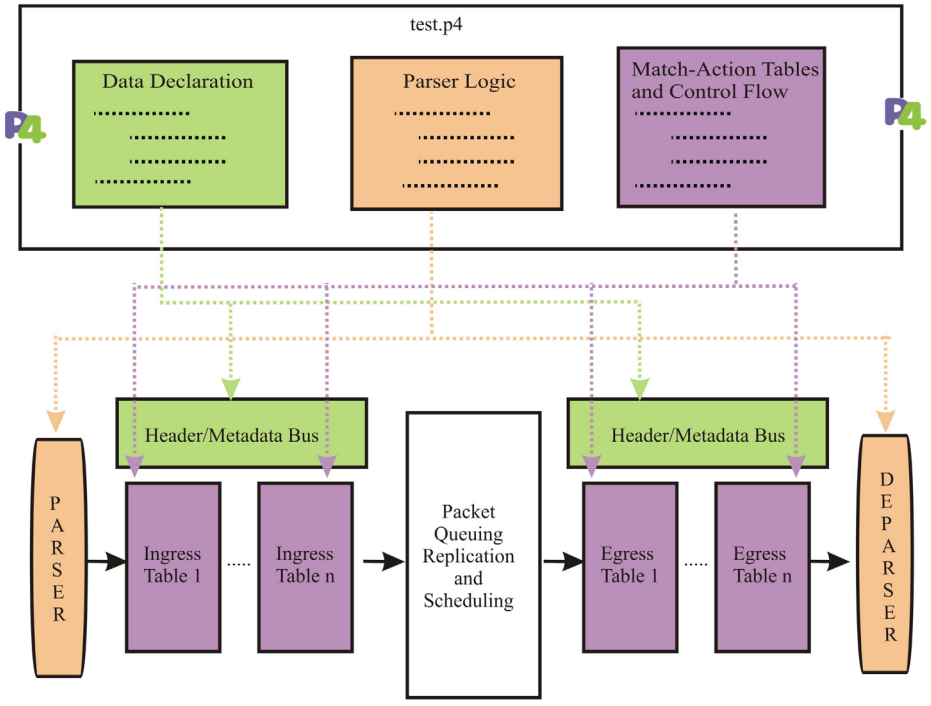

Fig. 3. Mapping of P4 code sections to the forwarding model [37].

2.5. P4 Runtime

P4 Runtime is an API that is used for the communication between the control plane and data plane [42]. The main benefit of P4 Runtime is to allow the controller to control any data plane regardless of whether it is built from fixed or programmable ASIC, FPGA, and NPU. It is independent of the features and protocol the data plane supports, therefore the same API can be used to control a large number of devices. Moreover, it does not depend on the placement of control plane; the control plane could be running in the local switch operating system or it could be a remote control plane like in the case of SDN. Sometimes, there is bit confusion between P4 runtime and P4 language because both contain the word "P4". The P4 language is used to define the behavior of data plane means what fields to match upon and what

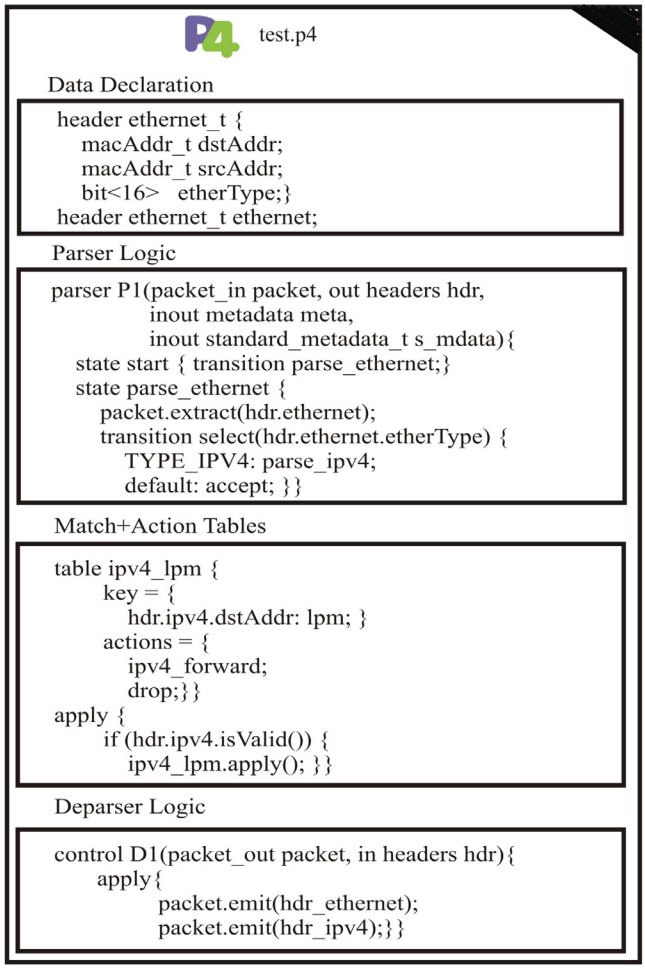

Fig. 4. P4 code sections [43].

actions it should perform. P4 Runtime is used by the control plane to control the data plane switches whose behavior is already specified by the P4 language.

3. Research efforts with P4 programmable data planes

In this section, we have discussed different research areas where P4 can really help. The recent research efforts in the data plane programmability indicates that P4 is really beneficial for SDN networks for efficient network monitoring, early detection of DDoS attacks, effective load balancing, and packet aggregation and disaggregation, without introducing the data-control plane communication overhead and without requiring any additional hardware.

3.1. Network monitoring

According to the report of Cisco Visual Networking Index: Forecast and Methodology (2016–2021) [45], in 2016, global IP traffic was 1.2 ZB per year or 96 EB (one billion Gigabytes [GB]) per month. By 2021, global IP traffic will reach 3.3 ZB per year, or 278 EB per month. The global IP traffic in 2021 is equivalent to 127 times of the IP traffic in 2005. In addition, due COVID-19 pandemic, there is a 25 to 35 percent jump in the internet traffic pattern in mid-march, 2020 from Feb, 2020 as governments implemented worldwide self-quartine and work-from-home policies [46]. With the drastic increase in internet traffic, DDoS and other web application attacks are also increasing, according to the report Q1 2020 [47].

To ensure the security of the network, we need more effective mechanism to collect network statistics for the network troubleshooting and the detection of attack traffic. SDN provides a flexible mechanism for the network monitoring, because logically centralized SDN controller maintains the global view of the network. The controller sends flow statistics request to SDN switches at specified time intervals for collecting the network information. However, it increases the bandwidth consumption of data-control plane communication channel that may cause network performance degradation [48]. To resolve this issue, P4 allows a network traffic monitoring at programmable switches, which significantly reduced the data-control communication overhead without requiring any additional hardware support [49]. Here, we have done the in-depth analysis of the research papers that used P4 language for network traffic monitoring (Table 4).

Li et al. [50] introduced encoded flowsets in which the flows and their counters are encoded at switches to ensure constant processing time for each packet. The size of encoded flowsets is small, therefore it takes less time to send this information to the remote controller. Then, network-wide decoding is performed at the controller to decode the flowsets across switches. It reduced the memory consumption to monitor the flows than the NetFlow monitoring tool. Liu et al. [51] proposed UnivMon that combined the advantages of sampling and custom sketching solutions. The programmable switches in the data plane process the incoming packets and implement the sketching algorithm in P4. The UnivMon controller collects the sketches from all switches and computes a network-wide estimation.

Sivaraman et al. [52] implemented the pipeline of hash tables to maintain the counters for large traffic flows, while removing the lighter flows over time. Pereira et al. [53] proposed the secure network monitoring framework by modifying the original count-sketch algorithm. To achieve this, original hash functions are replaced by the cryptographic hash functions that take as input 128-bits key. It becomes very difficult for an attacker to brute-force in an attempt to create hash collisions as the keys is not known to it. The results show that algorithm introduce fewer performance penalties in terms of latency and throughput as compared to original count-min sketch algorithm.

Harrison et al. [54] implemented the distributed monitoring system using the programmable switches for detecting the heavy flows in the network. In this approach, edge switches maintain the counter and threshold value for each key (Source IP and Destination IP). When the counter exceeds the local threshold value, then the switch sends the report containing both the counter and key value to the controller. The controller aggregates the counter value for the same key from the reports received from multiple switches and compares it with the global threshold value. If the total count exceeds the global threshold value, then it announces that key as a heavy flow. Afek et al. [55] proposed a Sample&Pick algorithm for detecting the large flows in the SDN network. Firstly, the sampling is performed on the flows passing through the SDN switch and sends these samples to the controller for identifying the suspicious large flows. Once the suspicious flows are identified, then controller inserts the rules into the switch to maintain the exact counter of these flows. By doing this, the controller gets the accurate count of heavy flows without installing too many rules into the switch.

Huang et al. [56] proposed SketchLearn algorithm that automatically learn the statistical properties of resource conflicts to monitor the massive amount of traffic with limited resources. It applied the statistical modeling to identify the hash collisions instead of pre-calculating the minimum resource requirement to achieve the sufficient accuracy. The statistical modeling makes this framework self-adaptive for various monitoring tasks (heavy hitter detection, heavy change detection, entropy estimation, flow size distribution, Cardinality estimation, and per-flow frequency) and requirements (small memory, fast packet processing, real time response, and generality). Martins et al. [57] proposed BitMatrix, that used the two probabilistic data structures bitmap and counter array sketches to monitor the multi-tenant networks in

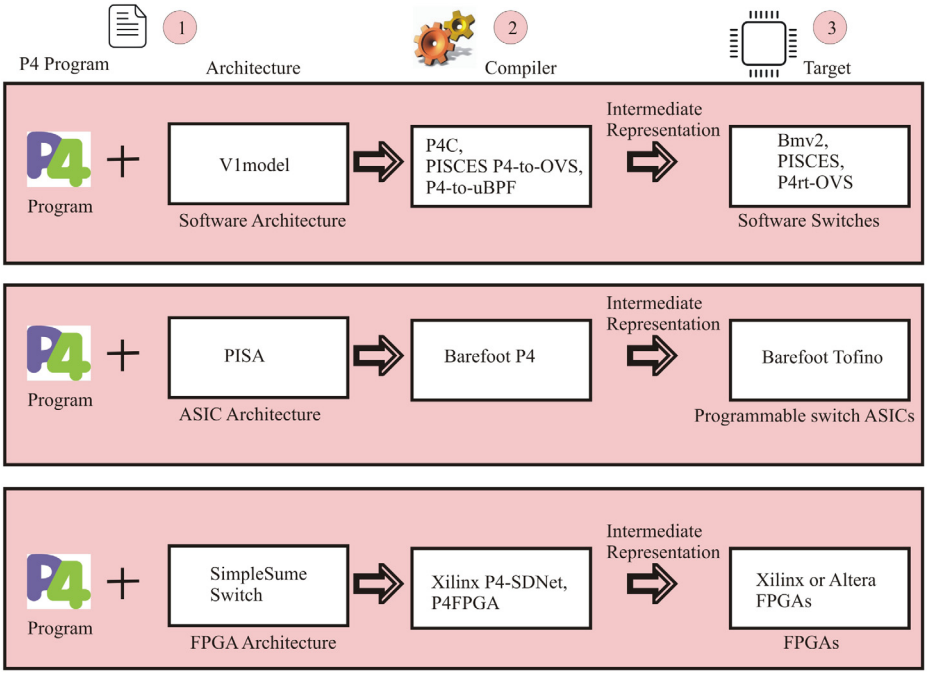

Fig. 5. P4 architectural models for various targets [44].

the P4 programmable data plane. By using the bitmaps and counter array, BitMatrix estimate the number of bytes and packets send to each tenant. The central server is responsible for collecting and comparing the data structure information from the P4-enabled switches and uses this information to identify how many packets and bytes are transferred in the specific path for each tenant.

In another related study [49], authors have implemented the P4-based network monitoring and scheduling framework in the SDN switches without the involvement of the controller. In this framework, monitoring module collects the status information (hop latency, queue occupancy, ingress port ID, switchID) of each switch in the forwarding plane through the In-band network telemetry (INT) framework. The traffic scheduling module gets the link status information from the monitoring function and schedule the traffic to the other available path on the occurrence of congestion. This method monitored the network status information without introducing additional delays and extra probe packets. Silva et al. [58] proposed IDEAFIX to identify the elephant flows in the IXP networks by taking the advantage of P4-based data planes. In this algorithm, edge switch extracts the 5-tuple features from each packet and identified the flow to which the packets belong to and stores the updated size and duration of flow in registers. Then, it compares this information with the predefined threshold. The notification is send to the controller after the detection of elephant flows. The controller reduces the effects of elephant flow on the entire network according to the quality of service policies.

Guan and Shen [59] proposed FlowSpy, that equally assign the monitoring task among the switches in the data plane by solving the Integer Linear Programming (ILP) problem. Hark et al. [60] presented an approach to filter out the irrelevant monitoring information on the P4 programmable data plane to save the computational and network resources of the controller. To achieve this, linear regression is implemented on switches that decide the importance of measurements. This method achieves sufficient accuracy while reducing the monitoring cost of forecast updates.

Suh et al. [61] proposed the Flexible sampling-based in-band network telemetry (FS-INT) to reduce the protocol overhead in original INT(O-INT) with sufficient accuracy. They implemented two sampling strategies: Rate-based (FS-INT(R)) and Event-based (FS-INT(E)) . In FS-INT(R), INT header is inserted in into every Rth packet instead of inserting it in all packets. In FS-INT(E), the value for the specific INT metadata type is inserted based on the predefined threshold value. The results show that FS-INT significantly reduced the protocol overhead as compared to O-INT.

Ding et al. [62] improved the distributed heavy hitter detection solution proposed by Harrison et al. [54] to deploy it in the network consisting of both fixed-function and programmable switches. This is the first study that deploys the monitoring solution in the hybrid network. In this approach, fixed-function switches are replaced by the programmable switches in an incremental way with the goal of monitoring highest number of distinct flows.

3.1.1. Discussion

Traditional network monitoring solutions such as SFlow [63], NetFlow [64], and SNMP [65] use the sampling techniques to collect the network information. While these techniques are suitable for collecting coarse-grained metrics they do not provide high accuracy unless they used the high sampling rates which is undesirable because it introduced high computation and communication overhead at the control plane. Hence, existing solutions run at low sampling rates to reduce the data-control plane communication overhead but they cannot perform an accurate monitoring of the network. To resolve this issue, recent research studies leverage the programmable data planes that divide the monitoring task between the remote controller and switches to track all the flows without sampling in a short interval of time. However, the majority of the proposed monitoring solutions [49,51–53,55–58,60,61] implemented at a single programmable switch, but, due to the limited information available on the single switch, some heavy hitter may be undetected or wrongly detected by the above approaches. Moreover, some monitoring solutions [49,51,53,58,60] require high computational resources that may result in performance bottleneck at the data plane switch.

A few research studies [50,54,59] performed the network-wide monitoring to provide an accurate picture of the network. However, in these approaches, same packet is counted multiple times at different switches and this duplicate information is not detected by the controller. In addition, they introduced high memory overhead to store the count and threshold for each key. To reduce the memory consumption, count-min sketch algorithm can be used that store this information only for the keys with counts greater than some minimum

Table 3
Various P4 compiling tools, supported targets and architectures.

P4 compiling tool	Supported targets	Supported architectures	Supported P4 version	Authorship	Description
P4C compiler [66]	BMv2 (Mininet Emulation tool, NS4 Simulation) [40,67]	V1Model [68], SimpleSumeSwitch [69], P4QueueDisc [70]	$P4_{14}$, $P4_{16}$	P4 Community	It is a reference compiler that provides standard frontend and midend. Its modularity makes the addition of new target specific backend easy. It uses p4c-bm2-ss backend.
Xilinx P4-SDNet compiler [71]	NetFPGA SUME board [39]	SimpleSumeSwitch [69]	$P4_{14}$	Industry	It converts the P4 code into Verilog module and allows the programmer to define their own extern functions.
Barefoot Capilano [72]	Barefoot Tofino [41]	Protocol Independent Switch Architecture (PISA) [44]	$P4_{14}$, $P4_{16}$	Industry	It is optimized to run at full line-rate on the PISA device and provides automatic debugging features.
P4-XDP compiler [73]	Netronome Agilio SmartNIC [74]	V1Model [68], XDP model [75], Portable Switch Architecture (PSA) [76]	$P4_{16}$	Industry	P4 compiler backend targeting XDP, the eXpress Data Path. XDP is designed for users who want programmability as well as performance. it is an extension of p4c compiler.
PISCES P4-to-OVS Compiler [77]	PISCES software switch [77]	V1Model [68]	$P4_{16}$	Academia	It generates new parse, match and action code in C for OVS and manipulates IR to optimize processing time and latency.
P4-to-uBPF compiler [78]	P4rt-OVS switch [79]	ubpf_model [80]	$P4_{16}$	Academia	It translates P4 program into C representation which is further compiled to BPF by using C language. It provides extern functions that are not directly supported by P4.
P4FPGA compiler [81]	Xilinx or Altera FPGAs [81]	SimpleSumeSwitch [69]	$P4_{14}$, $P4_{16}$	Academia	Extended version of p4c compiler with custom backend for generating FPGA code.

Table 4
Network monitoring using P4-Programmable data planes.

Ref	Objective	Technique used	Software/Hardware used	Results	Strengths	Weaknesses
Li et al. 2016 [50]	Monitor all the flows without sampling in short time scale	FlowRadar:encoded flowset data structure	NS-3 simulator, BMv2 Switch	• FlowRadar reduced the memory required to monitor 100K flows by 5.6% than NetFlow monitoring tool. • It requires only 2.3Gbps bandwidth per switch to send monitoring information of 100K flows to the controller with 10ms time scale.	• Monitor all the flows in cheap switches with low memory overhead and send the monitoring information to the controller in short time scale, • Scalable to large network because memory and bandwidth consumption do not change with more switches.	• Duplicate flow information collected from multiple switches is not discarded by the controller. • Flow decode time increases in a large network.
Liu et al. 2016 [51]	Network Flow monitoring with both generality and high accuracy	Universal Sketching	BMv2 Switch, Opensketch	• Reduced the data-control plane communication and memory consumption up-to three orders of magnitude • Memory consumption: 600 KB • Error gap between UnivMon and OpenSketch is < 3.6%	• Combine the advantages of both sampling and custom sketching solutions: generality and high accuracy • Comparable performance with respect to fixed-function switches while gaining a remarkable flexibility and programmability	• Large memory consumption at data plane • Communication overhead for sending a sketch to the controller at a specified time interval
Sivaraman et al. 2017 [52]	Identified the flow with large traffic volume	HashPipe: Heavy-Hitter detection algorithm	BMv2 Switch	• 15% less false negative than sample and hold method and 3%–4% less as compared to count-min sketch • memory consumption: 80 KB	Achieve high accuracy within the memory constraints of switches	• Consider only the flow size, did not take into account the duration of heavy flows • Introduced communication overhead
Pereira et al. 2017 [53]	Secure network monitoring in P4 programmable switches	Secure Count-Min Sketch algorithm	Mininet, BMv2 Switch	• Average latency overhead 10% • Latency difference between secure count-min sketch and original count-min sketch is 40 microseconds for sketch with 1 line and 160 microseconds for sketch with 20 lines	Secure than count-min sketch algorithm	• Performance penalties in terms of latency and through put • Computational resource consumption at data plane
Harrison et al. 2018 [54]	Distributed monitoring for heavy hitter detection	Threshold monitoring	Barefoot Tofino chipset	Reduced the communication overhead by 70%	• Reduced the overhead of communication channel without compromising accuracy. • Adaptive threshold value	• Duplicate counting of packets increase the memory consumption at switches and downgrade the monitoring performance • Do not consider the network consists of both programmable and legacy switches.
Afek et al. 2018 [55]	Detected heavy flow in the SDN switch	Sample and Pick algorithm	OpenSketch, NoviKit hardware switch	Error Rate: 3.3% , Memory usage: 2KB , controller-switch traffic: 220KB/s	Reduced the data-control plane communication and efficiently used the flow table	Do not perform the distributed monitoring of traffic at multiple switches
Huang et al. 2018 [56]	Monitoring of all the packets in fixed-size data structure	SketchLearn: a sketch-based measurement framework	Barefoot Tofino chipset, OpenvSwitch	• Memory usage:64KB, Per-Packet overhead:92 CPU, cycles, Throughput (Tofino switch): 99.96%, OVS DPDK:99.99%, OVS:99.93%. • Accuracy with memory size 32KB and 64KB is 98.4% and 100% respectively. Relative error with memory size 32KB and 64KB is 2.64% and 2.23% respectively.	• It achieves good accuracy while addressing the issues of approximate monitoring strategies. • High performance as there is no dependency between the counters in different levels that allows parallel processing.	Updating multi-level counter is time consuming. Due to this, it consumes various architecture-specific hardware resources to increase the performance.

(*continued on next page*)

Table 4 (*continued*).

Ref	Objective	Technique used	Software/Hardware used	Results	Strengths	Weaknesses
Martins et al. 2018 [57]	Multi-tenant monitoring	BitMatrix: bitmaps and counter array	Mininet, BMv2 Switch	Count the number of packets per tenant and number of packets on the specified path between two tenants	Monitoring for multi-tenant networks with adjustable accuracy	New sketches should be included to get more information about the networks
Geng et al. 2018 [49]	Network Monitoring and traffic scheduling	In-band network telemetry (INT) in data plane using P4	Mininet, BMv2 Switch	• packet loss rate 0%,0.98%,18% at 5M/s, 6M/s,7M/s UDP traffic rate • After rescheduling the traffic hop latency and queue occupancy close to zero	Avoid congestion by traffic rescheduling	Performance bottleneck at Bmv2 switch
Silva et al. 2018 [58]	Identified elephant flows in P4-based IXP networks	IDEAFIX	Mininet, BMv2 Switch, Ryu Controller	• Accuracy : 95% detection time: 0.4 ms • monitoring data overhead: 25KB as compared to traditional SDN having 17MB monitoring overhead	Early detection (0.40ms) of elephant flows	• Constant threshold value • More memory space is required in programmable switches for hash mapping • Increased the packet processing time by 30% than the traditional processing
Guan and shen 2019 [59]	Load balancing network monitoring	FlowSpy: assign monitoring task by solving ILP problem	Mininet, BMv2 Switch	With the increase in monitoring capacity, the completeness rates of monitoring tasks are stable that is near 100%	Evenly distribute the monitoring tasks among the switches in the data plane	It suffers from synchronization overhead of multiple switches.
Hark et al. 2019 [60]	Pre-process network monitoring information in data plane	Linear regression for estimating importance of measurements	Mininet, BMv2 Switch	when threshold is 500K, then measurements with small improvements are not forwarded to the controller that significantly reduced the monitoring cost	Achieve sufficient accuracy while significantly reducing the monitoring cost	Computational resource usage arises while requesting large number of statistics
Suh et al. 2020 [61]	Flexible sampling-based INT (FS-INT)	Sampling methods: Rate-based (FS-INT(R)) and Event-based (FS-INT(E))	BMv2 Switch	• Protocol overhead in O-INT (9%, 17%, 25.5%), FS-INT(R) (2.5%, 4%,6.5%), and FS-INT(E) (4%, 6.5%, 10%) for different path length 10, 20, and 30 respectively. It shows that FS-INT significantly reduced the protocol overhead • Average hop latency in FS-INT(E) (10.03 ms) is closer to the O-INT (10.16 ms) as compared to FS-INT(R) (9.15 ms). It shows that accuracy of FS-INT(E) is larger than FS-INT(R).	Reduced protocol overhead with sufficient accuracy.	• Predefined threshold value is used for sampling. • Did not use the real dataset in the experiment which makes their work unrealistic.
Ding et al. 2020 [62]	Distributed monitoring in a network comprises both fixed-function and programmable switches	Replace the legacy switches crossed by the highest number of distinct flows	Barefoot Tofino chipset, Mininet	• Comparable F1 score (0.823), less communication overhead (13898) and memory occupation (13799) with relative to state-of-the-art solution having F1 score (0.821), communication overhead (71877), memory occupation (760042). • Controller response time 1.5 ms	• Resolved the issue of partial deployment of programmable switches in a network • It avoids the duplicate counting of packets by multiple switches for efficient network-wide heavy-hitter detection.	Increased the processing overhead in the programmable switches as more read and write operation need to performed in the register than the existing solution.

Table 5
Detection of DDoS attacks using P4-Programmable data planes.

Ref	Objective	Technique used	Software/Hardware used	Results	Strengths	Weaknesses
Afek et al. 2017 [82]	Designed anti-DDoS system in SDN	Four anti-spoofing methods (HTTP Redirect, TCP Proxy, TCP Reset, TCP safe Reset),DNS anti-spoofing	Mininet, BMv2 Switch, OpenVswitch, POX Controller	• Without mitigation, web requests fail at 2.7K pps, with mitigation, web requests fail above attack rate 206K pps • RTT of TCP ping: 10micro second without mitigation, 33 micro seconds with mitigation	Significantly reduced the data-control plane communication overhead	Introduced significant delay for the legitimate requests
Korpershoek et al. 2018 [83]	Checked the suitability of p4 for the detection of DDoS attacks	History-Based IP filtering (HIF)	Mininet, BMv2 Switch	Drop 99.87% DDoS traffic and 10.19% normal traffic	High detection accuracy	It does not work if the attacker first generate regular traffic and then attack traffic
Datta et al. 2018 [84]	Designed a virtual firewall using programmable data plane	P4Guard: a software firewall written in P4 language	BMv2 Switch, CloudLab	Results are compared with VNGuard firewall, CPU usage in P4Guard:0.3%,VNGuard:46.8%	Platform-agnostic and protocol-independent firewall	Did not work for all size of packets
Paolucci et al. 2019 [85]	Apply P4 in multi-layer edge nodes	Stateful Traffic engineering and DDoS cyber security	NetFPGASume board, BMv2 Switch	Only 5 μs switch latency was experienced while running the P4 DDoS detection code	Comparable performance with respect to fixed-function switches while gaining a remarkable flexibility and programmability	Constant threshold value
Febro et al. 2019 [86]	Detection of DDOS attacks close to the source	Distributed source-based defense solution	BMv2 Switch	• Attack detection time: 1.47 s. • CPU usage: 71% during attack period and 24.49% in non-attack period, Memory usage peaked at 3%.	• It saves the computational and network resources by detecting the attack traffic at first hop. • Legitimate traffic not suffered during the attack period.	• Did not differentiate flash and attack traffic. • Constant threshold value.
Simsek et al. 2019 [87]	DDoS detection and mitigation in SDN	Hashing of source IP and Mac address, Two Rate Three Color Marker	BMv2 Switch	• CPU usage less than 50% during the attack period. • RTT increase after 500 packets/s due to the increase of CPU load on switches	• Detect attack close to their origin • Minimize delay experienced by the legitimate traffic	• More CPU consumption at switches • Cannot detect sophisticated attack pattern • Did not differentiate flash and attack traffic
Lapolli et al. 2019 [88]	DDoS detection in programmable data planes	Entropy based on SrcIP, DstIP	BMV2 switch	Accuracy: 98.2%, latency: 250ms	Reduced detection latency	Consumed TCAM memory to store pre-computed values or estimation in Match-Action table
Dimolianis et al. 2020 [89]	Detection of DDOS attacks using P4-enabled hardware	Multi-feature DDoS attack detection	Netronome Agilio CX SmartNIC	• In the underscaled attack, accuracy of two feature case (F2) is more as compared to three feature case (F3) due to small of attack traffic (5% of legitimate traffic). • In the Booter and overscaled attack, accuracy of F3 is more than F2 as it removes more false positive due to the addition of traffic symmetry feature.	• Differentiate between heavy benign and attack traffic. • Identified the victim of DDoS attacks.	• Cannot detect low-rate DDoS attacks. • Constant threshold value. • Mitigation of DDoS attacks is not considered.

size. Moreover, all the above monitoring solutions implemented in the network comprise only programmable switches. The major challenge for a successful adoption of these solutions is the deployment issue as it is impossible to replace all the existing fixed-function switches with the programmable switches. Ding et al. [62] did the best work in the field of network monitoring as they have considered the partial deployment scenario. Moreover, they have implemented the network-wide heavy hitter detection by collecting the information from multiple programmable switches. By doing this, it achieves the comparable accuracy by replacing less than half of the programmable switches as compared to the state-of-the art solution. This algorithm avoids the duplicate counting of packets and improves the monitoring performance. However, this approach introduced more packet processing time in P4-Programmable switches than the existing state-of-the-art solutions. This work can be extended to cover more monitoring tasks such as heavy change detection or entropy estimation. While above research works performed the distributed traffic monitoring, they do not provide a synchronized and consistent view of the network at short time scale. However, presenting an accurate picture of the network is essential to debug network faults such as loops, load imbalance, attack detection, etc.

3.2. Detection of DDoS attacks

With the rapid increase in internet traffic, there is a significant increase in both the quality and quantity of Distributed Denial-of-Service (DDoS) attacks. Due to the COVID-2019 pandemic, life has shifted almost entirely on the internet because people from worldwide are now studying, working, shopping, and having fun online that was never happen before. According to the report of Kaspersky [47], the number of DDoS attacks in Q1 2020 is double against the Q4 2019, by 80% against Q1 2019. However, most of the existing DDoS detection solutions are based on the packet sampling and transmission of gathered data to external software, which make it difficult to achieve high accuracy, less latency and low resource usage. In addition, the architecture of most promising technology SDN is also suffering from DDoS attacks due to the lack of intelligence in the SDN data plane [90]. The programmability of data plane has emerged as a promising solution as it allows forwarding devices to run DDoS detection algorithms and perform network traffic monitoring at line rate. The following research articles used the P4 language for the detection of DDoS attacks in SDN data plane without the involvement of SDN controller (Table 5).

Afek et al. [82] implemented the network anti-spoofing system by using the OpenFlow1.5 and P4 domain-specific language in the SDN data plane. They have proposed the 5 algorithms: HTTP Redirect, TCP Proxy, TCP Reset, TCP safe Reset, DNS anti-spoofing. In this approach, programmable switch act as a SYN proxy and forward only those requests to the server who completed the three-way handshake process. In another work [83], authors have checked the suitability of the P4 language for detecting the DDoS attack traffic by implementing the History-Based IP filtering algorithm. In this algorithm, database of legitimate sources is maintained in which they have stored the number of packets received from the particular source and the number of days a source has been seen. Then, the number of packets and days are compared with predefined threshold and the values that are below the threshold are dropped. Datta et al. [84] proposed P4Gurad that consists of two parts: P4Guard controller and data plane. The P4Gurad controller is responsible for collecting flow statistics from the p4-enables switches for the dynamic monitoring of network traffic. The P4 data plane processes the packets according to the firewall rules. The results show that P4Guard has better performance as compared to VNGuard that is a virtual firewall in NFV.

Paolucci et al. [85] provided the solutions for stateful traffic engineering and DDoS attack detection without the intervention of the controller. In the stateful traffic engineering, traffic offloading and optical bypass are implemented on the programmable data plane. To defend against DDoS port scanning attack, two register variables are maintained in which first variable store the TCP port of the previous packet of same session and second variable stored the number of attempt matching the attack pattern. If the number of attempts exceeds the threshold value, then it generates an attack alert. Febro et al. [86] implemented the distributed source-based DDoS defense solution to detect the attack close to the origin. This mechanism saved the computational and bandwidth resources by detection the malicious traffic at first hop. In this approach, every port of the P4-enabled edge switches counts the number of packets sent by the host connected to that port. To detect the attack, P4 controller compares the counter value at each port with the threshold (10 packets per second from one host). After exceeding the threshold, controller sends the command to P4 switch to drop all subsequent packets from the same ingress port. Simsek et al. [87] mitigate DoS attacks in the data plane switch that calculates the hash value of source IP and MAC address and compare it with the previous stored hash value.

If it does not match and timestamp of the last attack and current packet is less than 5 s then attack is detected. Lapolli et al. [88] implemented the entropy based DDoS detection mechanism in the programmable switches. As the P4 does not provide any support for arithmetic logarithmic functions; therefore it estimates the entropy value using longest-prefix match (LPM) table. Moreover, to meet the strict time and memory constraints of forwarding devices, it approximates the frequencies of distinct IP address using the count sketch data structure.

Dimolianis et al. [89] implemented a multi-feature DDoS defense solution in the P4-enabled hardware devices. They have considered the three traffic features to detect the DDoS attacks: (i) total number of flows received by the monitored subnets in a given interval of time (ii) percentage of flows received by the particular subnets out of total flows (iii) ratio of incoming flows to outgoing flows. They considered two cases for evaluation: two feature case (F2) that contains only first two features and three feature case (F3) that contains all three features. In addition, three attack scenarios are considered: underscaled that contains small amount of attack traffic, Booter trace that contains original attack traffic in the dataset, and overscaled that contains 5 times of the original attack traffic. Results show that accuracy of F2 is higher than the F3 in underscaled attack and accuracy of F3 is higher than F2 in other two scenarios.

3.2.1. Discussion

Due to the lack of intelligence in the data plane switches in traditional SDN architecture, network traffic monitoring and DDoS attack detection is performed at the controller. However, it consumes computational and bandwidth resources of the controller and introduced a significant delay for the DDoS attack detection because switches need to send the packets to the controller for filtering normal and attack traffic. To resolve these issues, some researchers have implemented the DDoS defense solutions in data plane switches using the P4 language. Existing P4-based DDoS defense solutions [82–88] used the simple algorithms (TCP Proxy, history-based IP filtering, Entropy based on source and destination IP, hashing) for the detection of DDoS attacks. The implementation of sophisticated DDoS defense solutions may be challenging in P4 because it does not provide any support for floating point arithmetic, logarithmic function, and loops. In addition to it, the above DDoS defense solutions did not differentiate flash and attack traffic and are not able detect the low-rate DDoS attacks.

Beside this, the majority of the defense solutions [82,84–87] used the local traffic generator tools to generate the normal and malicious traffic. However, these traffic generators are not able generate traffic as in the benchmark data sets. In addition, some of the information theory based defense solutions [85,86,89] used the static threshold value for the anomaly detection which makes their work unrealistic. Moreover, the above approaches identified the occurrence of an attack without identifying the source of the attack. Furthermore, there is a need to

Table 6
Load balancing using P4-Programmable data planes.

Ref	Objective	Technique used	Software/Hardware used	Results	Strengths	Weaknesses
Katta et al. 2016 [91]	Load balancing using P4-Programmable switches	HULA (Hop-by-hop Utilization-aware Load balancing Architecture)	NS-2	1.6 to 3.3 times better flow completion time as compared to existing load balancing algorithm (CONGA) at 50% and 90% network load	HULA only maintains the congestion state for best next hop instead for entire path to destination, therefore it is more scalable than CONGA approach [92].	• It used probe packets that consume the bandwidth and downgrades the entire network performance. • Not designed for Multipath TCP(MPTCP)
Benet et al. 2018 [93]	Load balancing for MPTCP protocol	MP-HULA: multipath transport aware load balancing	NS-2 Simulator	Reduced flow completion time from 2.9 to 3.4 times than HULA at 80% and 50% load.	Utilized the features of MPTCP to improve the network performance.	• As in HULA, probe packets are used to track congestion information that results in bandwidth consumption. • This work can be extended to consider different flow priorities.
Ye et al. 2018 [94]	Load balancing using P4-Programmable switches for data centers	W-ECMP load balancing	Mininet, Bmv2 Switch	W-ECMP achieves 10% improvements in flow completion time as compared to ECMP and 0.7% improvements as compared to HULA	• Increase network status update speed. • Efficient utilization of the link bandwidth	Did not improve FCT for large flows as compared to HULA
Pit-Claude et al. 2018 [95]	Stateful Layer-4 Load Balancing	SilkRoad framework that perform load balancing using switch ASICs	BMv2 switch	For processing the same amount of traffic in switches consume 1/500 of power and 1/250 of capital cost as compared to software load balancer	• Reduced power and capital cost by two orders of magnitude. • Simultaneously load balance ten million connections.	• Large memory consumption for storing connections state in ASICs. • On the failure of SilkRoad switch, the connections that use old DIP pool may break PCC.
Miao et al. 2017 [96]	Load-aware load balancing	SHELL: load aware stateless load balancer	10G NetFPGA-SUME	Throughput: 59.8 Mpps Worst-case latency: 8.96 μs	Less than 1% of the connection lost	Did not compare the results with the state-of-art load balancing solutions
Hsu et al. 2020 [97]	Dynamic load balancing across multiple paths in the programmable data plane	Hash based data structure for adaptive weighted traffic splitting	NS-3 simulator, BMv2 Switch	• Reduced flow completion time by 22.1% and 16% as compared to HULA and ECMP load balancing strategies at 80% workload. • Less load imbalance (20%) as compared to ECMP technique (80%)	• Traffic splitting based on the path weights is done at line rate. • High packet processing throughput. • Less number of packets are required to update path weights.	• It delivers packets of same flows over multiple paths, which may cause out-of-order packet delivery. • For certain type of workload, operator may prefer traffic splitting , but for other type of workload, splitting may not be required. It does not provide this flexibility to operator.
Zhang et al. 2020 [98]	Load balancing in a virtual switch without require changes to the network fabric	VMS: a packet level load balancing	Pronto3295 48-port switch, NS-3 Simulator	• In the symmetric topology, it reduced the flow completion time 47% and 22% as compared to ECMP and CONGA respectively. • In the asymmetric topology, it reduced the flow completion time up to 3 times and 1.4 times than PerPacket and CONGA approach.	• Resolve the out-of-order packet arrival problem by using a re-sequence buffer at receiver side. • Did not require changes to the network fabric. • Reduced the load imbalance when there is a topology asymmetry.	• Introduced the 4% latency overhead to re-ordering the packets at receiver side. • It does not consider the non-TCP traffic for load balancing.

implement the defense solution in every data plane switch as part of DDoS detection. In these solutions, each switch maintains the local view of the network; therefore it cannot provide a complete picture of the network traces. Hence, the implementation of detection mechanism with the complete picture of the network traces is the major research challenge. Some of the above defense solutions [87,88] consumed TCAM memory to store pre-computed values or estimation in match-action table of data plane switches. Therefore, the implementation of DDoS defense solutions in programmable switches without increasing the complexity and cost is an open research issue. In future, DDoS defense solutions based on Machine Learning, Artificial Neural Network, Information theory based, and Signature based IDS with publicly available datasets can be implemented in the data plane switches for the precise identification of attacks.

3.3. Load balancing

In current data centers, load balancing is usually implemented in software servers. The software load balancers are responsible for mapping the connection to the same server, even if the DIP pool changes or load is distributed differently across the servers [95]. However, it requires thousands of servers or 3.75% of data center size. Moreover, it introduced high latency and jitter. In contrast, load balancing in high-performance switching ASICs can process same amount of traffic with less power and capital cost. Moreover, offloading load balancing to switching ASIC improves throughput and latency of the network. Table 6 shows the various load balancing solutions using the P4-Programmable data planes. Katta et al. [91] proposed HULA (Hop-by-hop Utilization-aware Load balancing Architecture), written in the P4 language to implement scalable load balancing for programmable switches. This algorithm used the special probe packets to get the link utilization information from the network. However, this method works on the best path selection which makes it easy to congest the best path. In addition, HULA used the probe packets to get the congestion information that consumes the link bandwidth and downgrade the entire network performance.

Miao et al. [95] proposed SilkRoad framework that implemented the stateful layer-4 load balancing to provide high throughput and low latency. In this approach, Per Connection Consistency (PCC) is maintained, so that all the packets having same connection must be forwarded to the same Destination IP (DIP). To resolve the issue of storing a million of connections simultaneously with the limited amount memory in switches, the small hash of connection is stored instead of 5-tuples in match fields. This solution ensures PCC and support DIP pool updates with low latency and low cost and is able to store 10 million connections simultaneously is hardware switches. To solve the problems of HULA load balancing scheme [91], Benet et al. [93] proposed the MP-HULA that instead of selecting the one best path, it selects the k best paths to reduce the congestion on one link. Moreover, it utilized the Multipath TCP (MPTCP) protocol to improve the network performance.

Another related study proposed W-ECMP (weighted Equal-cost multipath) algorithm [94] that select a path with weighted probability. In addition, it encapsulates the congestion information in normal traffic that increase update frequency in high network load. In another work [96], authors have proposed SHELL, an application aware stateless load balancer for the programmable hardware. It maintains the history table that enables the P4 load balancer to direct all subsequent packets of a flow to the application server handling the connection. The results show that it achieves attainable performance while providing application-aware load balancing.

Hsu et al. [97] present a novel data structure that achieves load-aware traffic splitting in the P4-enabled switches. This study is different from the WCMP-based approaches as it addresses the challenges due to hardware constraints of programmable switches. The WCMP algorithm take a long time to update the path weights as it replicate the hash table entries based on the weight of each path. This study efficiently updates the path weights by assigning a unique region in the output space of hash function according to the weight of each path. Instead of modifying the large number of table entries, now the updates require modification only in the region boundaries. In this way, it takes less time to update path weights as compared to WCMP-based algorithms. Zhang et al. [98] implemented a Virtual Multi-channel Scatter (VMS), a flexible load balancing solution that requires no changes in the network fabric. In this approach, TCP packets are classified over multiple paths based on the available bandwidth on each path. It uses the virtual window size to represent the available bandwidth of each path. It reduced the out-of-order packet delivery problem by using re-ordering buffer at the receiver side. Moreover, it significantly reduced the load imbalance when there is a topology asymmetry. Results shows that it significantly reduced the flow completion time over ECMP and CONGA approach. However, it suffers from latency overhead due to use of re-sequence buffer.

3.3.1. Discussion

Majority of the above load balancing solutions [91,93–96] evenly distributed the load in symmetric topology, but it introduces load imbalance when there is a topology asymmetry. In some solutions [93,97], packets of same flows are transmitted over multiple paths that may lead to out-of-order packet delivery. In addition, some algorithms [91, 93] used the probe packets to track congestion information that may cause bandwidth consumption of communication channel and network performance degradation. Beside this, the design of above load balancing solutions raise various issues due to the hardware constraints of programmable switches which include a limited number of per-packet processing operation at each stage, limited number of stages in the programmable pipeline, and accessing a register memory of one stage from another stage. Moreover, a large memory is required in the switches to store the connections state information [95].

A recent work [98] that addresses the load imbalance and out-of-order packet delivery problems but it suffers from little overhead at receiver side. Therefore, there is still a need for implementing an efficient load balancing solution that addresses the challenges due to the hardware constraints of programmable switches. All the above solutions improve the network performance by implementing the distributed load balancing in programmable switches. However, the practical deployment of these solutions in a datacenter environment introduced a high replacement cost because it requires the replacement of all switches with programmable once. In the future, researchers can implement cost-effective load balancing solution in which instead of replacing all the switches, portion of switches to be replaced with programmable switches to realize distributed load balancing in multi-rooted topologies. Moreover, the implementation of end-host driven load balancing in programmable switches is another research issue in which end hosts request more or less traffic to be forwarded towards them according to the availability of resources. Therefore, there is still a need for implementing an efficient load balancing solution that addresses the challenges due to the hardware constraints of programmable switches.

3.4. Packet aggregation and disaggregation

This section discussed the research efforts that utilize the data plane programmability to perform the packet aggregation and disaggregation in the SDN switches. Packet aggregation is an efficient method to send small packets in the network. Due to the rapid development of IoT, it is forecast that a large number of IoT devices will be deployed in various environments for different IoT applications. For instance, smart water/electricity/gas metering, in which large quantity of smart meters send their usage reports to the central office for billing and energy saving purposes. The usage reports sent to the central office are short, that make most of the IoT packets very small in size. However, the packet headers of the small IoT packet occupy a large portion of

Table 7
Packet aggregation and disaggregation using P4-Programmable data planes.

Ref	Objective	Technique used	Software/Hardware used	Results	Strengths	Weaknesses
Lin et al. 2018 [99]	IoT packets aggregation using programmable data planes	Packet header manipulation in P4 switch	Barefoot Tofino chip, BMv2 switch	Packet aggregation is performed at 100Gbps line rate	No extra time is required for the packet aggregation	• Processing time to disaggregate a large packet comprises N IoT packets is same as the processing N individual IoT packets. • Aggregate fixed number of small IoT packets ($N = 8$) with same payload size.
Wang et al. 2019 [100]	Packet aggregation and disaggregation to provide high throughput	Packet header and payload manipulation in the pipeline of hardware P4 switches	100BF-65X Tofino chip	• Maximum throughput of packet aggregation process is 100Gbps i.e. the line rate of P4 switch. • Maximum throughput of packet disaggregation cannot reach 100Gbps under the different parameter settings.	Provides the highest packet aggregation throughput reported so far	• It does not work for variable size of IoT packets. • Introduced long delay for collecting the enough packets to start the aggregation process. • Performance bottleneck at the disaggregation process.
Wang et al. 2020 [101]	Packet aggregation and disaggregation of small packets with different payloads	Packet header and payload manipulation in the pipeline of hardware P4 switches	Barefoot Tofino chip	Maximum throughput of both aggregation and disaggregation process is 100Gbps	• Aggregate small packets with different payload size. • Number of packets to be aggregated may also vary according to the payload size.	• This study optimizes only single metric of performance (throughput). However, applications may have different performance requirements based on its characteristics. • Homogeneous network environment is considered. • They have taken the fixed number of IoT devices.
Madureira et al. 2020 [102]	IoT packets aggregation using programmable data planes	IoTP: implement packet aggregation in P4-enabled switches	Mininet, BMv2 Switch	• Increase network efficiency by 78% and 5 times faster than the End-to-End strategy). • Energy anatomy of IoT devices in End-to-End strategy is more as compared to IoTP.	• Improve network efficiency and reduce average delay experienced by the packet aggregation process. • Dynamically improve the packet aggregation algorithms by considering the network information and underlying communication technology. • IoTP can support multiple aggregation algorithms.	There are some performance limitations (maximum memory limit) of Mininet emulation environment that limit the amount of data stored in a single packet by the IoTP. Instead of Mininet tool, real IoT devices can be used to analyze the performance of IoTP in hardware-based data aggregation.

total packet. When a large amount of small IoT packets delivered in the network, then the IoT packet headers consume the large percentage of network bandwidth. Packet aggregation reduces the significant amount of bandwidth consumption by using same header for multiple aggregated packets [100]. In addition, packet aggregation in pipelines of switch ASIC can achieve greater throughput as compared to existing approaches. Table 7 shows the various packet aggregation approaches using the P4-Programmable data planes.

Lin et al. [99] performed the packet header manipulation in the P4-enabled switches to aggregate the small IoT packets into a large packet. With this approach, payload of IoT packet is considered as a header because P4-enabled switches can manipulate the header of the packets. After storing the payloads of N IoT packets in register, it considered these payloads as header fields to creates the aggregated payload and pass it to the second switch. Then, second switch performs the disaggregation of large packet. Wang et al. [100] implemented the packet aggregation and disaggregation in the P4-enabled hardware switches to provide the maximum throughput.

In this approach, there is a need to modify the packet payload for the packet aggregation. When an IoT packet is received by the P4 switch, then the parser in the switch extract the headers from the packet and check the type flag. If the type flag indicates that the packet is an IoT packet, then it is aggregated according to the packet aggregation process. If the switch receives an aggregated packet then it disaggregated according to the resubmit and clone-based disaggregation process.

In another related study [101], the above work is extended to perform the aggregation of small packets with variable size of payloads in the P4-enabled switches. In this approach, the number of packets to be aggregated depends on the size of packet payload. It performs the packet aggregation only when the sufficient packets payloads are available to make 1500 byte aggregated packet (byte-counter > 1467 bytes). During the disaggregation process, switch cloned the aggregated packet, and recirculated it N-1 times to extract N-1 small packets and last small packet is extract from original copy. This mechanism provides the 100Gbps throughput both in aggregation and disaggregation process. Madureira et al. [102] implemented the Internet of Things Protocol (IoTP) using the P4 language to perform the packet aggregation in data plane switches based on the network information such as link bandwidth, delay, MTU, and underlying communication technologies (802.11, BLE, ZigBee). In this approach, IoTP devices aggregate the data that belong to the same IoT application. As IoT headers provide the network information, therefore IoTP can be used with other aggregation solutions to improve the network performance. Results show that it improves the network efficiency by 78% and 5 times faster than the End-to End strategy.

3.4.1. Discussion

In IoT scenarios, packet aggregation and disaggregation is performed within the IoT devices which introduced long delay and low throughput. Besides, IoT devices required more computational resources and presents high power consumption to perform the packet aggregation. In addition, software based solutions have been proven to be inefficient as compared to hardware solutions in term of packet switching speed and power consumption. To resolve these issues, recent research studies [99–102] embed the packet aggregation algorithms in P4-enabled hardware switches to reduce the overall network delay. All the above solutions achieved the maximum throughput (100Gbps) during the packet aggregation process. However, some of the works [99,100] suffered from a performance bottleneck during the disaggregation process as processing time required to disaggregate a large packet comprises N IoT packets is same as the processing N individual IoT packets. Besides this, they statically defined the number of packets to be aggregated and assumed all packets have a same payload size which is not practically feasible. This problem is solved by [101] that aggregate variable number of packets with different payload size.

Moreover, the majority of the solutions [99–101] do not take into account the underlying network details such as link bandwidth, delay, and communication technology involved (802.11, ZigBee, and BLE). Moreover, they considered the homogeneous network environment means packets are sent by the single application. In addition, only single performance metric is considered for optimization but applications may have different performance requirements. To resolve all these issues, Madureira et al. [102] did the best work in this field. They considered the underlying network information to dynamically improve the packet aggregation algorithms. The SDN controller is responsible to select the packet aggregation algorithm based on the underlying network status and requirements of IoT applications. This study validates the proposed approach in the Mininet emulation environment. However, the performance limitations of the emulation environment affect the data aggregation process. In the future, researchers can implement sophisticated aggregation solutions with real IoT devices to further improve the efficiency of the network.

4. Research gaps

After the critical analysis of existing research articles in the field of P4-Programmable data plane, we identified a number of research gaps. These gaps will further help in the development of P4-based solutions for efficient network monitoring, DDoS detection, load balancing, and packet aggregation and disaggregation.

- Majority of the researchers [49–52,57–61,82–84,86,87,93–95,97,102] evaluate their work in BMv2 switch, which is not a production ready switch because it does not support protocols like OpenConfig, gNMI, and gNOI that are used for configuration, monitoring, and operations respectively. In addition, the processing capacity of BMv2 switch is very less as compared to the real hardware switches. Therefore, future researchers can perform their experiments in a production ready environment like Stratum based switches [103] which support all the above protocols and work in the hybrid SDN environment.
- The existing P4-based DDoS defense solutions [82–88] did not differentiate between the flash and attack traffic as they assume that variation in the normal and attack traffic is sufficient enough for DDoS attack detection. In addition, attackers may perform low-rate DDoS attacks to bypass the defense solutions. So, the detection of low-rate DDoS attacks using the P4-Programmable data planes is also a major research issue.
- Most of the DDoS defense solutions [82,84–87] used the synthetic data sets generated by the local traffic generating tool. The non-availability of benchmark data sets containing both malicious and normal traffic is another research issue.
- Some of the research works [57,83–86,89,97] used a small topology for the experiments which makes their validation unrealistic.
- Some authors [95,96] used P4 for implementing load balancing algorithms in switch ASICs, but it requires large memory for storing the connections states in ASIC. So, the implementation of efficient load balancing solutions with the limited memory in the data plane switches is another research issue.
- We found that majority of the network monitoring solutions [49–53,55,57,58,60] require high computational resources that may result in performance bottleneck at data plane switches. Only a few studies [54,59,62] distribute the monitoring tasks among data plane switches for avoiding the overloading of the network monitoring capacity of each switch. Although, these solutions provide better network performance, but they suffer from synchronization and communication overhead. So, the distributed monitoring in programmable data planes, while keeping the synchronization and communication overhead at a minimum is a major research issue that needs to be addressed.

- Majority of the network monitoring solutions [49–55,57–60,62] are too expensive to be enabled continuously due to the computation and communication overhead at the controller. Therefore, reporting of flow statistics is enabled only when there is a problem in the network. However, it may cause missing of some information which may be necessary to find the root-cause of the problem. Therefore, research is required to identify the problem at runtime and send only that information to the controller which is critical to debug the root-cause of problems in a network.
- All the above solutions [49,51–53,55,57–60,82–85,87,88,91,94–96,100,101] did not use the popular open source SDN controllers such as Ryu [104], OpenDaylight [105], ONOS [106]to control the P4 programmable data planes. So, the integration of P4 programmable data planes with the SDN controllers is a major research issue because it can ease the migration of existing SDN applications to the combined SDN model with programmable data planes and enables the centralized management of P4-Programmable data planes.
- Although P4-based solutions reduced the computation and communication overhead at the control plane, but they may increase the complexity in data plane switches. Therefore, efficient deployment of P4-based algorithms in switches while minimizing the complexity is an open research issue.
- Most of the existing research studies [49–61,82–89,91,93–102] work only in the programmable switches. However, the major challenge for a successful adoption of these solutions is the deployment issue as it is impossible to replace all the existing fixed-function switches with the programmable switches. Therefore, the integration of programmable switches with the fixed-function switches is still an open research issue.

5. Research challenges

In spite of the growing interest in the programmability of the data plane, its widespread usage depends on the advances in a number of issues like data plane security, availability of P4 programmable switches and migration cost, P4 design issues, packet processing speed, and implementation of stateful network functions.

5.1. Data plane security

The concept of programmable networks is not a new because Active networking [107] is the first research effort in the direction of programmable networks in the late 1990, which did not get widespread adoption due to the lack of security. Similarly, there are some security challenges in the P4-Programmable data planes. Firstly, software is more prone to errors as compared to the hardware. Moreover, data plane forwarding behavior is defined by the end user, which is normally less experienced and careful as compared to the vendor [31]. Secondly, an attacker can exploit the programmability feature to modify the forwarding behavior of the device which creates new attack vectors. Therefore, assertions to be added to the programs and verification should be performed for improving the security of the programmable data plane.

5.2. Availability of P4-Programmable switches and migration costs

P4 is still a relatively young technology and there are only a few manufactures that make P4-Programmable Ethernet switch ASICs (Barefoot Tofino, Broadcom Tomahawk). Moreover, the cost of replacing fixed-function devices with the programmable one is higher than adding one or more traditional device in the existing network [108]. In addition, P4 experts are needed to define the behavior of programmable switches. The integration of SDN and traditional network is still a challenge for the research community [26]. Moreover, the addition of programmable data planes in the SDN network raises the network complexity level at a peak point. To solve all these problems, the incremental deployment is a feasible solution in which programmable and fixed-function SDN devices should be incrementally deployed in the traditional network.

5.3. P4 design issues

The domain-specific language (P4) does not include floating point numbers. In addition, due to the lack of loop constructs, the P4 is unable to process the packet payload in an efficient way [83]. Moreover, there is no support for pointers, references, and dynamic memory allocation. These language limitations make it difficult to implement efficient algorithms which require deep packet inspection such as DDoS detection, network monitoring. For instance. Lapolli et al. [88] implemented the DDoS detection algorithms by estimating the value of entropy using longest prefix match tables instead of computing the exact entropy, because P4 does not support the arithmetic logarithmic functions.

Therefore, adding the above language constructs is an open research challenge for the research community.

5.4. Packet processing speed

It seems strange that how programmable data planes can process the data at line speed. This observation comes into mind because protocol-agnostic switches may increase processing overheads leading to undesirable packet delays. But recent developments such as Protocol-independent Switch Architecture (PISA) [109] are trying to address this problem. It utilizes the Reconfigurable Match Tables (RMT) concept for changing data plane packet processing logic without the need of hardware up-gradation. Moreover, programmable switch ASICs such as Barefoot Tofino, Broadcom Tomahawk, operate at terabits per second. In spite of these advances, research community should put intention on packet processing speed while considering the limitations of programmable switches such as limited amount of memory, types of supported operation, and computation allowed on each packet.

5.5. Implementation of stateful network functions

In Network Function Virtualization (NFV), programmable switches are considered as a best target point for the deployment of network functions such as load balancers, intrusion detection systems, and network address translators because of their in-network location and high-speed processing rates [110]. However, due to the limited memory available on programmable switches, it becomes difficult to implement memory-intensive network functions that require stateful packet processing such as IDS, stateful load balancers and firewalls. To resolve this issue, one possible solution is that switch ASIC vendors can extend the memory capacity (adding more SRAM or off-chip DRAM). It will incur high cost and does not provide flexibility and scalability. Another possible solution is based on the observation that in data centers, most of network resources and DRAM on servers are under-utilized. By using the DRAM of servers, programmable switches can extend their memory capacity with low cost. However, there are some technical challenges such as performance, fault-tolerance, and load balancing to implement this solution in practice.

5.6. Automated testing and formal verification of P4-Programmable data planes

While data plane programmability provide immense benefits such as rapid innovation, protocol development, in-band network telemetry, it also raises challenges related to the correctness of data plane. However, there are various issues that arise while building the verification tool for P4 data planes. One issue is that P4 does not fully define the behavior of data plane which make it difficult to perform the verification without

the involvement of the control plane. For example, P4 only defines the structure of match-action tables, the flow rules of match-action tables are installed by the control plane at runtime. One way to resolve this problem is to delay the verification until the flow rules are known. However, it changes the verification from compile time to run time and explicit the need to repeatedly perform the verification when flow rules are changed which would be expensive. Another way is to allow the P4 programmer to define the control plane interface using symbolic constraints. However, it requires programmers to write the control plane interface by hand. Hence, automated synthesis of control plane interfaces is a promising future research direction. Another issue that is related to data plane verification is scalability because P4 language constructs like match-action tables have dense conditional structure. Due to this, standard verification techniques like symbolic execution, which travel across all control flow paths are not able to scale well. Therefore, developing a scalable verification tool which avoids explicit traversal of all control flow paths is an open research challenge.

6. Conclusion and future directions

This paper provides the comprehensive review of P4-Programmable data planes along with its strengths over the traditional SDN architecture. We discussed all the technical aspects related to P4-Programmable data planes i.e. various P4 compilers, Architectures, Runtime API, and Targets. Moreover, we critically reviewed the prominent research articles that indicate P4 is really beneficial in SDN networks for efficient network monitoring, DDoS attack detection, load balancing, packet aggregation and disaggregation, without introducing the data-control plane communication overhead and additional hardware.

After the in-depth analysis of research articles related to network monitoring, it has been found that distributed network monitoring is ignored by most of the researchers and need to be addressed properly. As far as the detection of DDoS attacks is concerned, the implementation of sophisticated defense solutions based on machine learning, artificial neural network, and generalized information theory metric is still an open research issue. A few studies exist that performed packet aggregation and disaggregation in a programmable data plane. As packet aggregation is highly beneficial for IoT network for delivering the small packets, so the researchers should focus on this field who want to use SDN in IoT networks. In P4-based load balancing solutions, the implementation of effective load balancing algorithms in switch ASICs requires large memory for storing the connections states that is another research issue. In addition, we observed that most of issues exist due to the limited computational resources available in programmable switches and design issues in the P4 language. Beside this, more than 60% of the prominent research articles validated their work in emulation and simulation environment using small topology for the experiments which makes their validation unrealistic.

As a part of our future work, we are motivated to implement effective DDoS defense solution in the combined SDN model with the P4 programmable data plane. In the SDN architecture, both network traffic monitoring and DDoS attack detection is performed at the controller due to the lack of intelligence in the SDN switches. It introduces data-control plane communication overhead, performance bottleneck at the controller, and introduced a long delay for the detection of DDoS attacks because every packet require to be sent to the controller for deep packet inspection. To resolve these issues, we will divide the task of network traffic monitoring and DDoS detection between the control plane and data plane. By leveraging the data plane programmability, SDN switches will be responsible for network monitoring and send the alert message to the controller. The DDoS detection will be performed at the controller machine because effective DDoS defense solutions based on the machine learning require high computational resources. At the end, we conclude that network community should put more intention on the programmability of data plane to make the SDN architecture more flexible and programmable. In the next few years data-plane programmability will become commonplace and P4 plays an important role in it.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Diego Kreutz, Fernando M.V. Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, Steve Uhlig, Software-defined networking: A comprehensive survey, Proc. IEEE 103 (1) (2014) 14–76.

[2] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, Jonathan Turner, Openflow: enabling innovation in campus networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.

[3] Hyojoon Kim, Nick Feamster, Improving network management with software defined networking, IEEE Commun. Mag. 51 (2) (2013) 114–119.

[4] Michael Jarschel, Simon Oechsner, Daniel Schlosser, Rastin Pries, Sebastian Goll, Phuoc Tran-Gia, Modeling and performance evaluation of an openflow architecture, in: 2011 23rd International Teletraffic Congress (ITC), IEEE, 2011, pp. 1–7.

[5] Soheil Hassas Yeganeh, Amin Tootoonchian, Yashar Ganjali, On scalability of software-defined networking, IEEE Commun. Mag. 51 (2) (2013) 136–141.

[6] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al., P4: Programming protocol-independent packet processors, ACM SIGCOMM Comput. Commun. Rev. 44 (3) (2014) 87–95.

[7] PL Vision, P4 Language, https://plvision.eu/expertise/sdn-nfv/p4, Accessed: 2020-06-20.

[8] Simon Jouet, Richard Cziva, Dimitrios P. Pezaros, Arbitrary packet matching in openflow, in: 2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR), IEEE, 2015, pp. 1–6.

[9] Hamid Farhady, HyunYong Lee, Akihiro Nakao, Software-defined networking: A survey, Comput. Netw. 81 (2015) 79–95.

[10] Kamal Benzekki, Abdeslam El Fergougui, Abdelbaki Elbelrhiti Elalaoui, Software-defined networking (SDN): a survey, Secur. Commun. Netw. 9 (18) (2016) 5803–5833.

[11] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, Haiyong Xie, A survey on software-defined networking, IEEE Commun. Surv. Tutor. 17 (1) (2014) 27–51.

[12] Shiva Rowshanrad, Sahar Namvarasl, Vajihe Abdi, Maryam Hajizadeh, Manijeh Keshtgary, A survey on SDN, the future of networking, J. Adv. Comput. Sci. Technol. 3 (2) (2014) 232–248.

[13] Jacob H. Cox, Joaquin Chung, Sean Donovan, Jared Ivey, Russell J. Clark, George Riley, Henry L. Owen, Advancing software-defined networks: A survey, IEEE Access 5 (2017) 25487–25526.

[14] Yosr Jarraya, Taous Madi, Mourad Debbabi, A survey and a layered taxonomy of software-defined networking, IEEE Commun. Surv. Tutor. 16 (4) (2014) 1955–1980.

[15] Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, Thierry Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, IEEE Commun. Surv. Tutor. 16 (3) (2014) 1617–1634.

[16] Qiao Yan, F. Richard Yu, Qingxiang Gong, Jianqiang Li, Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges, IEEE Commun. Surv. Tutor. 18 (1) (2015) 602–622.

[17] Ijaz Ahmad, Suneth Namal, Mika Ylianttila, Andrei Gurtov, Security in software defined networks: A survey, IEEE Commun. Surv. Tutor. 17 (4) (2015) 2317–2346.

[18] Sandra Scott-Hayward, Sriram Natarajan, Sakir Sezer, A survey of security in software defined networks, IEEE Commun. Surv. Tutor. 18 (1) (2015) 623–654.

[19] Nasrin Sultana, Naveen Chilamkurti, Wei Peng, Rabei Alhadad, Survey on SDN based network intrusion detection system using machine learning approaches, Peer-to-Peer Netw. Appl. 12 (2) (2019) 493–501.

[20] Izzat Alsmadi, Dianxiang Xu, Security of software defined networks: A survey, Comput. Secur. 53 (2015) 79–108.

[21] Heng Zhang, Zhiping Cai, Qiang Liu, Qingjun Xiao, Yangyang Li, Chak Fone Cheang, A survey on security-aware measurement in SDN, Secur. Commun. Netw. 2018 (2018).

[22] Murat Karakus, Arjan Durresi, A survey: Control plane scalability issues and approaches in software-defined networking (SDN), Comput. Netw. 112 (2017) 279–293.

[23] Fetia Bannour, Sami Souihi, Abdelhamid Mellouk, Distributed SDN control: Survey, taxonomy, and challenges, IEEE Commun. Surv. Tutor. 20 (1) (2018) 333–354.

[24] Murat Karakus, Arjan Durresi, Quality of service (QoS) in software defined networking (SDN): A survey, J. Netw. Comput. Appl. 80 (2017) 200–218.

[25] Jochen W. Guck, Amaury Van Bemten, Martin Reisslein, Wolfgang Kellerer, Unicast QoS routing algorithms for SDN: A comprehensive survey and performance evaluation, IEEE Commun. Surv. Tutor. 20 (1) (2017) 388–415.

[26] Rashid Amin, Martin Reisslein, Nadir Shah, Hybrid SDN networks: A survey of existing approaches, IEEE Commun. Surv. Tutor. 20 (4) (2018) 3259–3306.

[27] Yash Sinha, K. Haribabu, et al., A survey: Hybrid sdn, J. Netw. Comput. Appl. 100 (2017) 35–55.

[28] Xinli Huang, Shang Cheng, Kun Cao, Peijin Cong, Tongquan Wei, Shiyan Hu, A survey of deployment solutions and optimization strategies for hybrid SDN networks, IEEE Commun. Surv. Tutor. 21 (2) (2018) 1483–1507.

[29] Alaitz Mendiola, Jasone Astorga, Eduardo Jacob, Marivi Higuero, A survey on the contributions of software-defined networking to traffic engineering, IEEE Commun. Surv. Tutor. 19 (2) (2016) 918–953.

[30] Ian F. Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, Wu Chou, A roadmap for traffic engineering in SDN-OpenFlow networks, Comput. Netw. 71 (2014) 1–30.

[31] Weverton Luis da Costa Cordeiro, Jonatas Adilson Marques, Luciano Paschoal Gaspary, Data plane programmability beyond openflow: Opportunities and challenges for network and service operations and management, J. Netw. Syst. Manage. 25 (4) (2017) 784–818.

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

[33] Enio Kaljic, Almir Maric, Pamela Njemcevic, Mesud Hadzialic, A survey on data plane flexibility and programmability in software-defined networking, IEEE Access 7 (2019) 47804–47840.

[34] Pravein Govindan Kannan, Mun Choon Chan, On programmable networking evolution, CSI Trans. ICT 8 (2020) 69—76.

[35] Brandon Butler, What P4 programming is and why it's such a big deal for software defined networking, 2017, https://www.networkworld.com/article/3163496/what-p4-programming-is-and-why-it-s-such-a-big-deal-for-software-defined-networking.html, Accessed: 2020-06-28.

[36] Praveen Desu, Ram Subramoniam, DPDK based Fast Path Programing using P4, https://www.google.co.in/url?sa=i&url=https%3A%2F%2Ffast.dpdk.org%2Fevents%2Fslides%2FDPDK-2017-04-India-P4.pdf&psig=AOvVaw1c9Kv4hKtikr4RPR4kQpdr&ust=1594267430901000&source=images&cd=vfe&ved=0CAkQjhxqFwoTCPCBhpHjvOoCFQAAAAAdAAAAABAP. Accessed: 2020-07-02.

[37] Vladimir Gurevich, P4 tutorial, 2015, https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwidyqPRr73qAhWTSH0KHWzNCxwQFjACegQIAxAB&url=https%3A%2F%2Fp4.org%2Fassets%2FNov-2015-P4-Bootcamp-P4-Tutorial.pdf&usg=AOvVaw0z_W920_sNSbYTYK-tv0KW, Accessed: 2020-06-22.

[38] P4 Language Consortium, P4_16 Language Specification. https://p4.org/p4-spec/docs/P4-16-v1.2.1.html, Accessed: 2020-07-02.

[39] Noa Zilberman, Yury Audzevich, G. Adam Covington, Andrew W. Moore, Netfpga SUME: Toward 100 Gbps as research commodity, IEEE Micro 34 (5) (2014) 32–41.

[40] Behavioral model (bmv2), 2015, https://github.com/p4lang/behavioral-model, Accessed: 2020-06-28.

[41] Barefoot Tofino2, https://www.barefootnetworks.com/products/brief-tofino-2/, Accessed: 2020-06-21.

[42] Timon Sloane Nick McKeown, Jim Wanderer, P4 runtime, 2017, https://p4.org/api/p4-runtime-putting-the-control-plane-in-charge-of-the-forwarding-plane.html, Accessed: 2020-07-02.

[43] P4 language tutorials, https://github.com/p4lang/tutorials, Accessed: 2020-07-02.

[44] P4.org.

[45] Cisco Visual Networking Index, Cisco visual networking index: Forecast and methodology, 2016–2021, Complet. Vis. Netw. Index (VNI) Forecast 12 (1) (2017) 749–759.

[46] Tom Bienkowski, COVID-19 network traffic patterns: A worldwide perspective from our customers, 2020, https://www.netscout.com/blog/Network-Traffic-in-the-Age-of-COVID-19, Accessed: 2020-07-02.

[47] Alexander Gutnikov Oleg Kupreev, Ekaterina Badovskaya, Ddos attacks in Q1 2020, 2020, https://securelist.com/ddos-attacks-in-q1-2020/96837/, Accessed: 2020-07-02.

[48] Pang-Wei Tsai, Chun-Wei Tsai, Chia-Wei Hsu, Chu-Sing Yang, Network monitoring in software-defined networking: A review, IEEE Syst. J. 12 (4) (2018) 3958–3969.

[49] Junjie Geng, Jinyao Yan, Yangbiao Ren, Yuan Zhang, Design and implementation of network monitoring and scheduling architecture based on P4, in: Proceedings of the 2nd International Conference on Computer Science and Application Engineering, 2018, pp. 1–6.

[50] Yuliang Li, Rui Miao, Changhoon Kim, Minlan Yu, Flowradar: A better netflow for data centers, in: 13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16), 2016, pp. 311–324.

[51] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, Vladimir Braverman, One sketch to rule them all: Rethinking network flow monitoring with univmon, in: Proceedings of the 2016 ACM SIGCOMM Conference, 2016, pp. 101–114.

[52] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, Shan Muthukrishnan, Jennifer Rexford, Heavy-hitter detection entirely in the data plane, in: Proceedings of the Symposium on SDN Research, 2017, pp. 164–176.

[53] Fábio Pereira, Nuno Neves, Fernando M.V. Ramos, Secure network monitoring using programmable data planes, in: 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE, 2017, pp. 286–291.

[54] Rob Harrison, Qizhe Cai, Arpit Gupta, Jennifer Rexford, Network-wide heavy hitter detection with commodity switches, in: Proceedings of the Symposium on SDN Research, 2018, pp. 1–7.

[55] Yehuda Afek, Anat Bremler-Barr, Shir Landau Feibish, Liron Schiff, Detecting heavy flows in the SDN match and action model, Comput. Netw. 136 (2018) 1–12.

[56] Qun Huang, Patrick P.C. Lee, Yungang Bao, Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, 2018, pp. 576–590.

[57] Regis F.T. Martins, Fábio L. Verdi, Rodolfo Villaça, Luis Fernando U. Garcia, Using probabilistic data structures for monitoring of multi-tenant P4-based networks, in: 2018 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2018, pp. 00204–00207.

[58] Marcus Vinicius Brito da Silva, Arthur Selle Jacobs, Ricardo José Pfitscher, Lisandro Zambenedetti Granville, IDEAFIX: Identifying elephant flows in P4-based IXP networks, in: 2018 IEEE Global Communications Conference (GLOBECOM), IEEE, 2018, pp. 1–6.

[59] Bowei Guan, Shan-Hsiang Shen, Flowspy: An efficient network monitoring framework using p4 in software-defined networks, in: 2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall), IEEE, 2019, pp. 1–5.

[60] Rhaban Hark, Divyashri Bhat, Michael Zink, Ralf Steinmetz, Amr Rizk, Preprocessing monitoring information on the SDN data-plane using P4, in: 2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE, 2019, pp. 1–6.

[61] Dongeun Suh, Seokwon Jang, Sol Han, Sangheon Pack, Xiaofei Wang, Flexible sampling-based in-band network telemetry in programmable data plane, ICT Express 6 (1) (2020) 62–65.

[62] Damu Ding, Marco Savi, Gianni Antichi, Domenico Siracusa, An incrementally-deployable P4-enabled architecture for network-wide heavy-hitter detection, IEEE Trans. Netw. Serv. Manag. 17 (1) (2020) 75–88.

[63] Kostas Giotis, Christos Argyropoulos, Georgios Androulidakis, Dimitrios Kalogeras, Vasilis Maglaris, Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments, Comput. Netw. 62 (2014) 122–136.

[64] Shivam Choudhary, Bhargav Srinivasan, Usage of netflow in security and monitoring of computer networks, Int. J. Comput. Appl. 68 (24) (2013).

[65] William Stallings, SNMP, SNMPv2, SNMPv3, and RMON 1 and 2, Addison-Wesley Longman Publishing Co., Inc., 1998.

[66] P4 Language Consortium, et al., P4c: A new, alpha-quality reference compiler for the p4 programming language, 2019.

[67] Jiasong Bai, Jun Bi, Peng Kuang, Chengze Fan, Yu Zhou, Cheng Zhang, NS4: enabling programmable data plane simulation, in: Proceedings of the Symposium on SDN Research, 2018, pp. 1–7.

[68] V1model Architecture. https://github.com/p4lang/p4c/blob/master/p4include/v1model.p4, Accessed: 2020-06-28.

[69] SimpleSumeSwitch Architecture Model, https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiClP629LzqAhVCX30KHZ4SC3sQFjAAegQIBBAB&url=https%3A%2F%2Fcs344-stanford.github.io%2Flectures%2FLecture-3-dev-tools.pdf&usg=AOvVaw2Jaf3Paa_KrOnZG1_esefj, Accessed: 2020-06-28.

[70] P4 QueueDisc, https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjt7fCd77zqAhWazTgGHbALDdgQFjACegQIARAB&url=https%3A%2F%2Fcs344-stanford.github.io%2Flectures%2FLecture-3-dev-tools.pdf&usg=AOvVaw2Jaf3Paa_KrOnZG1_esefj, Accessed: 2020-06-22.

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

[72] Barefoot Networks, The World's Fastest & Most Programmable Networks, https://barefootnetworks.com/resources/worlds-fastest-most-programmable-networks/, Accessed: 2020-06-21.

[73] A p4 to xdp compiler back-end, 2018, https://github.com/vmware/p4c-xdp, Accessed: 2020-06-15.

[74] Netronome. P4 Programmability for the Netronome Agilio SmartNIC. https://www.netronome.com/blog/p4-programmability-for-the-netronome-agilio-smartnic/.

[75] P4 XDP model, 2019, https://github.com/vmware/p4c-xdp/blob/master/p4include/xdp_model.p4, Accessed: 2020-07-02.

[76] Portable switch architecture (psa), 2020, https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwj4p9yV9bzqAhUIbn0KHcNyDhYQFjAAegQIAxAB&url=https%3A%2F%2Fp4lang.github.io%2Fp4-spec%2Fdocs%2FPSA.pdf&usg=AOvVaw3qpKulP3T26SGVkMXKeRiI, Accessed: 2020-07-02.

[77] Muhammad Shahbaz, Sean Choi, Ben Pfaff, Changhoon Kim, Nick Feamster, Nick McKeown, Jennifer Rexford, Pisces: A programmable, protocol-independent software switch, in: Proceedings of the 2016 ACM SIGCOMM Conference, 2016, pp. 525–538.

[78] Tomasz Osiński, p4c-ubpf: a New Back-end for the P4 Compiler, https://p4.org/p4/p4c-ubpf, Accessed: 2020-06-15.

[79] P4rt-OVS: Programming protocol-independent, runtime extensions for open vswitch using P4, 2019, https://github.com/Orange-OpenSource/p4rt-ovs#p4rt-ovs-programming-protocol-independent-runtime-extensions-for-open-vswitch-using-p4, Accessed: 2020-06-22.

[80] P4 UBPF model, 2020, https://github.com/p4lang/p4c/blob/master/backends/ubpf/p4include/ubpf_model.p4, Accessed: 2020-07-02.

[81] Han Wang, Robert Soulé, Huynh Tu Dang, Ki Suh Lee, Vishal Shrivastav, Nate Foster, Hakim Weatherspoon, P4fpga: A rapid prototyping framework for p4, in: Proceedings of the Symposium on SDN Research, 2017, pp. 122–135.

[82] Yehuda Afek, Anat Bremler-Barr, Lior Shafir, Network anti-spoofing with SDN data plane, in: IEEE INFOCOM 2017-IEEE Conference on Computer Communications, IEEE, 2017, pp. 1–9.

[83] Jan-Jaap Korpershoek, Filtering DDoS traffic using the P4 programming language, in: 29th Twente Student Conference on IT, 2018, pp. 1–8.

[84] Rakesh Datta, Sean Choi, Anurag Chowdhary, Younghee Park, P4guard: Designing p4 based firewall, in: MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM), IEEE, 2018, pp. 1–6.

[85] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, P. Castoldi, P4 edge node enabling stateful traffic engineering and cyber security, J. Opt. Commun. Netw. 11 (1) (2019) A84–A95.

[86] Aldo Febro, Hannan Xiao, Joseph Spring, Distributed SIP ddos defense with P4, in: 2019 IEEE Wireless Communications and Networking Conference (WCNC), IEEE, 2019, pp. 1–8.

[87] Goksel Simsek, Hakan Bostan, Alper Kaan Sarica, Egemen Sarikaya, Alperen Keles, Pelin Angin, Hande Alemdar, Ertan Onur, Dropppp: A P4 approach to mitigating dos attacks in SDN, in: International Workshop on Information Security Applications, Springer, 2019, pp. 55–66.

[88] Ângelo Cardoso Lapolli, Jonatas Adilson Marques, Luciano Paschoal Gaspary, Offloading real-time ddos attack detection to programmable data planes, in: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), IEEE, 2019, pp. 19–27.

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

[90] Rochak Swami, Mayank Dave, Virender Ranga, Software-defined networking-based ddos defense mechanisms, ACM Comput. Surv. 52 (2) (2019) 1–36.

[91] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, Jennifer Rexford, Hula: Scalable load balancing using programmable data planes, in: Proceedings of the Symposium on SDN Research, 2016, pp. 1–12.

[92] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al., CONGA: Distributed congestion-aware load balancing for datacenters, in: Proceedings of the 2014 ACM Conference on SIGCOMM, 2014, pp. 503–514.

[93] Cristian Hernandez Benet, Andreas J. Kassler, Theophilus Benson, Gergely Pongracz, Mp-hula: Multipath transport aware load balancing using programmable data planes, in: Proceedings of the 2018 Morning Workshop on in-Network Computing, 2018, pp. 7–13.

[94] Jin-Li Ye, Chien Chen, Yu Huang Chu, A weighted ECMP load balancing scheme for data centers using P4 switches, in: 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), IEEE, 2018, pp. 1–4.

[95] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, Minlan Yu, Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics, in: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, 2017, pp. 15–28.

[96] Benoît Pit-Claudel, Yoann Desmouceaux, Pierre Pfister, Mark Townsley, Thomas Clausen, Stateless load-aware load balancing in p4, in: 2018 IEEE 26th International Conference on Network Protocols (ICNP), IEEE, 2018, pp. 418–423.

[97] Kuo-Feng Hsu, Praveen Tammana, Ryan Beckett, Ang Chen, Jennifer Rexford, David Walker, Adaptive weighted traffic splitting in programmable data planes, in: Proceedings of the Symposium on SDN Research, 2020, pp. 103–109.

[98] Yiran Zhang, Jun Bi, Zhaogeng Li, Yu Zhou, Yangyang Wang, VMS: Load balancing based on the virtual switch layer in datacenter networks, IEEE J. Sel. Areas Commun. 38 (6) (2020) 1176–1190.

[99] Yi-Bing Lin, Shie-Yuan Wang, Ching-Chun Huang, Chia-Ming Wu, The SDN approach for the aggregation/disaggregation of sensor data, Sensors 18 (7) (2018) 2025.

[100] Shie-Yuan Wang, Chia-Ming Wu, Yi-Bing Lin, Ching-Chun Huang, High-speed data-plane packet aggregation and disaggregation by P4 switches, J. Netw. Comput. Appl. 142 (2019) 98–110.

[101] Shie-Yuan Wang, Jun-Yi Li, Yi-Bing Lin, Aggregating and disaggregating packets with various sizes of payload in P4 switches at 100 Gbps line rate, J. Netw. Comput. Appl. (2020) 102676.

[102] André Luiz R. Madureira, Francisco Renato C. Araújo, Leobino N. Sampaio, On supporting IoT data aggregation through programmable data planes, Comput. Netw. (2020) 107330.

[103] Stratum, 2020, https://www.stordis.com/open-networking/network-operating-system/stratum/, Accessed: 2020-07-02.

[104] Saleh Asadollahi, Bhargavi Goswami, Mohammed Sameer, Ryu controller's scalability experiment on software defined networks, in: 2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC), IEEE, 2018, pp. 1–5.

[105] Jan Medved, Robert Varga, Anton Tkacik, Ken 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, IEEE, 2014, pp. 1–6.

[106] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al., ONOS: towards an open, distributed SDN OS, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, 2014, pp. 1–6.

[107] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, Gary J. Minden, A survey of active network research, IEEE Commun. Mag. 35 (1) (1997) 80–86.

[108] Gorby Kabasele Ndonda, Ramin Sadre, A two-level intrusion detection system for industrial control system networks using p4, in: 5th International Symposium for ICS & SCADA Cyber Security Research 2018 5, 2018, pp. 31–40.

[109] Andy Fingerhut Antonin Bas, P4 tutorial, 2018, https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwj5qaDX_brqAhWEj-YKHY9qCe4QFjABegQIARAB&url=https%3A%2F%2Fpc.nanog.org%2Fstatic%2Fpublished%2Fmeetings%2FNANOG75%2F1855%2F20190219_Bas_P4_Tutorial_v1.pdf&usg=AOvVaw2JEpMaadJgpEsbgYHSRn19, Accessed: 2020-06-19.

[110] Cheng-Liang Hsieh, Ning Weng, Virtual network functions instantiation on SDN switches for policy-aware traffic steering, in: Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems, 2016, pp. 119–120.

Sukhveer Kaur received the B.Tech. degree from Punjabi University Patiala and M.Tech. degree from Punjab Technical University Jalandhar in Computer Science Engineering. She is currently pursuing her Ph.D. in Information Technology from UIET, Panjab University Chandigarh. Her research interest is on Software-Defined Networking and Cloud Computing. She has published 20 papers in International & National Conferences. She has got 4 years teaching experience. She is UGC-JRF qualified.

Dr. Krishan Kumar is currently Professor in Department of Information Technology, University Institute of Engineering and Technology, Panjab University, Chandigarh. He has done B.Tech. Computer Science & Engineering from National Institute of Technology, Hamirpur in 1995. He completed his Master of Software Systems from Birla Institute of Technology & Sciences, Pilani in 2001. He finished his regular Ph.D. from Indian Institute of Technology, Roorkee in February, 2008. He has more than 20 years of teaching, research and administrative experience. His general research interests are in the areas of Network Security and Computer Networks. Specific research interests include Intrusion Detection, Protection from Internet Attacks, Web performance, Network architecture/protocols, and Network measurement/ modeling. He has published 2 national and 2 International Books in the field of Computer Science & Network security. He has published more than 120 papers in national/International peer reviewed/Indexed/impact factor Journals and IEEE, ACM and Springer proceedings. His publications are well cited by eminent researchers in the field.

Naveen Aggarwal is actively working in the area of Computer Vision and Data Mining. He did his Ph.D. from GGSIPU, Delhi in year 2011 and M. Tech. in Computer Science and Engineering from IIT, Kharagpur. Dr. Aggarwal has been awarded with Faculty Innovation award from the Infosys foundations. Different Effort Estimation Model developed by Dr. Aggarwal are appreciated and being used by Axede Corporation, USA and Birlasoft Corp. India. He has over 18 publications in International journals and 55 publications in international and national conferences. He has joined PU as Asstt. professor in Computer Science and Engineering at UIET in February 2005. Earlier, he has worked for three years from 2002 to 2005 in Punjab Engineering College, Chandigarh.