IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 20 April 2023, accepted 3 May 2023, date of publication 22 May 2023, date of current version 7 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3275756

SURVEY

A Survey on P4 Challenges in Software Defined Networks: P4 Programming

BHARGAVI GOSWAMI[1], MANASA KULKARNI[2], AND JOY PAULOSE[2]
[1]Department of Computer Science, Queensland University of Technology, Brisbane, QLD 4000, Australia
[2]Department of Computer Science, CHRIST (Deemed to be University), Bengaluru, Karnataka 560029, India

Corresponding author: Bhargavi Goswami (goswamib@qut.edu.au)

This work was supported by the Raja Jurdak and Queensland University of Technology.

ABSTRACT Software Defined Networking (SDN) has been a prominent technology in the last decade that increases networking programmability. The SDN philosophy decouples the application, control, and data plane to increase the network programmability. The data plane is an essential but unsolved component that receives less attention than control and application planes. Traditionally, the data plane uses fixed functions that forward packets using a limited number of protocols. The P4 (Programming Protocol-independent Packet Processors) language makes it possible to program SDN data plane, which push the SDN to the next level. In the research community and industry, programming the data plane has garnered significant attention. Surprisingly, there has been no comprehensive reviews of programmable data-plane switches, which have many advantages in today's networks. The authors reviewed the evolution of networks from legacy to programmable data planes, explained the fundamentals of programmable switches, and summarized the network generation from traditional to programmable networks. In this paper, SDN is described from a P4-centric standpoint and discusses over 75 related research papers. Several taxonomies for the field are provided, outline potential research areas, and provide greater details regarding the patterns that have led to the development of this technology.

INDEX TERMS Network programmability, P4 language, programmable switches, software-defined networks, data plane programmability.

I. INTRODUCTION

As the network structure grows in size, the network management needs to be simplified. Enterprise networks significantly influence working environments [1]. However, the demand for delivering a growing volume of data and applications in real-time is rapidly increasing. Network technologies have become an essential components for almost all human activities [2]. The number of devices on the Internet and the amount of traffic passing through them are growing exponentially. Owing to the complex alliances between specialized packet-forwarding hardware and operating systems, network administrators can manage network hardware using other software packages [3]. This can be a customized network that blocks the development of new protocols and services. Network devices must be customized for a particular activity to reduce engineering costs. However, there has been a push toward commodity and universal network devices [4].

Traditional networks are expensive to deploy and manage, and new-generation networks must overcome the rigidity of existing network architectures [5]. For large-scale organizations, the cumulative effect of rigidity can be disastrous concerning architectural investments. It can also take years to deploy new ideas, from design to simulation, testing, standardization, and installation of network equipment. Consequently, the network is at a critical stage and must be programmable [6].

Several technologies are used in these networks. Software-defined networks are fascinating technologies that encourage creativity in network management and design. Although SDN appears to have emerged rapidly, there has been a long history of attempts to make computer networks more programmable [7]. SDN allows network operators to create a custom network logic. SDN also enables device

The associate editor coordinating the review of this manuscript and approving it for publication was Chakchai So-In.

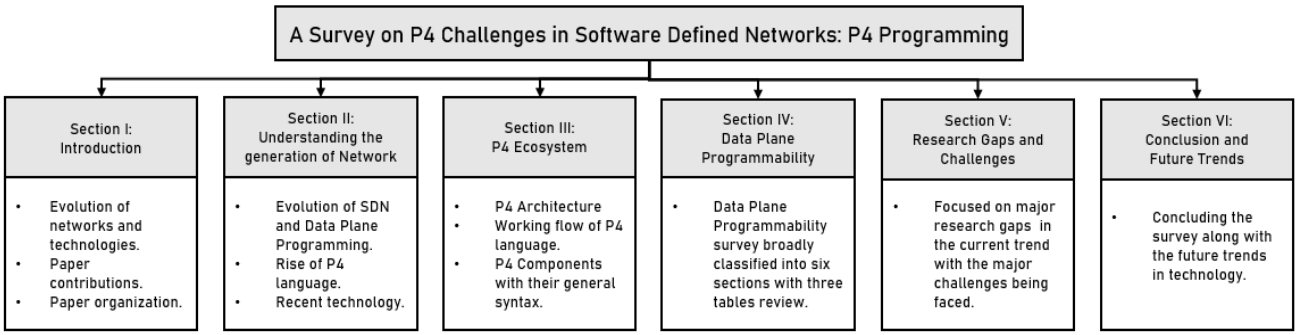

FIGURE 1. Paper Roadmap.

manufacturers to focus on designing well-defined building blocks instead of creating products for various client use cases [8]. A further benefit of programmability is that it frees device manufacturers from creating products for various client-use cases. Engineering resources can instead be devoted to designing well-defined building blocks that can be used to build logistics specific to a company [9].

Although programmable data-plane technologies have not become widely used and popular, numerous techniques still exist [10]. How can fundamental packet-processing primitives be modified and used to provide the broadest range of network applications with the best performance? How can the operator know the possible complex processing logic so that the configuration is quick, safe, and verifiable? How can we reproduce, monitor, and abstract the state of fleeting packet processing deeply ingrained by this logic? What are its most advantageous uses and applications? These are some of the problems that the networking communities are currently debating.

A. CONTRIBUTION

The following are the contributions of this paper that is exceptional and not available in state of art research:

- This paper discusses network generation from traditional to programmable networks.
- Introduces data plane programmability using P4 in detail with its architecture and P4 components.
- Discussion on the technical aspect of the P4 language.
- A thorough examination of P4 programmability on the SDN platform is provided.
- The research gaps and challenges are discussed in depth.

B. PAPER ORGANIZATION

This paper is organized into six sections, as provided in Figure 1, to understand the SDN and P4 connectivity during the tera-byte bandwidth era. Section II gives a detailed overview of the evolution of SDN and the rise of P4 in networking technology. Section III explains the aspects of P4, its Architecture, and its components in detail. Section IV details the data-plane programmability, and the tabular format on a few papers surveyed is presented. Section V discusses the research challenges and opportunities, and the final Section VI concludes the study along with future trends.

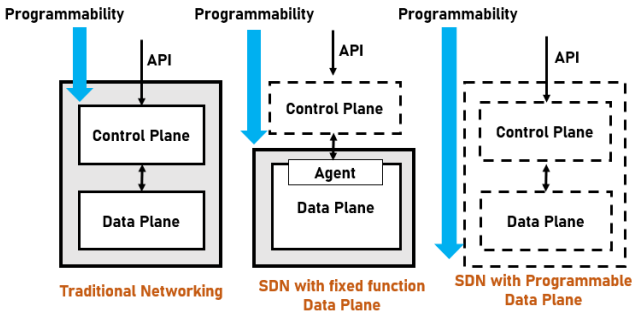

FIGURE 2. Traditional to programmable networking.

II. UNDERSTANDING THE GENERATION OF NETWORKS

Before diving deeper into the survey, we provide an overview of the various developments that have led to the need for data plane programmability. This section provides further details on how network generation has changed from traditional to programmable networks, as shown in Figure 2. Figure 2 is explained in-detail in the further sub-sections. Each subsection outlines the changes that caused the networks to become programmable. This section concludes with the importance of P4 in SDN.

A. TRADITIONAL NETWORKS

Traditional networks operate over fixed-function equipment, such as routers, switches, and application delivery controllers. SDN provides flexibility to the network and adapts it to the growing networks and security. Traditional networking involves using dedicated devices to perform specific tasks. Traditional networks have always faced challenges in terms of flexibility and are expensive to deploy and manage. Traditional networks are hardware-oriented and SDN are software-oriented. Vendors adopt SDN because traditional networking lacks flexibility and, most importantly, fails to adapt to the growing network and security. The control and data plane are tightly coupled and rigid to the functionalities, as shown in figure 2.

B. SOFTWARE DEFINED NETWORKS

SDN is the most prominent technology used by researchers to make networks programmable. SDN comprises Controller, Southbound, and Northbound APIs, and has been in the networking industry for more than ten years. SDN is a strategy executed when the network needs to be programmable [11]. The network control and data plane are physically separated, and the control plane can control multiple devices. The control plane is programmable with SDN built into it, as seen in figure 2 but the data plane is not flexible because programmability has not yet reached it. However, management and configuration are pushed to centralized consoles [12]. SDN has been widely adopted in Amazon, Facebook, and Google Data Centers [13]. The below sub-sections will take through the journey of SDN evolution and how did the data-plane layer was flexible.

1) OpenFlow

Once SDN stabilized, the OpenFlow protocol became popular and proved as of the most profound SDN protocols. The OpenFlow protocol is used to configure SDN-enabled devices. The OpenFlow protocol is a crucial component in the development of SDN solutions [14]. It acts as a communication protocol in the SDN controller to interact directly with network devices in the data plane [15]. All devices that communicate with the SDN controller must be compatible with the OpenFlow protocol [16]. Protocol standards have become increasingly complex in recent years. The SDN controller sends changes to the switch/router flow table via this interface [17], allowing network administrators to segment traffic, manage flows for optimal performance, and configure and apply new information [18].

2) SDN EVOLUTION

Table 1 shows that first, the researchers focused on transforming IT, as it was a challenging task it is termed ''IT Craftsmanship.'' It took two decades for IT Craftmanship to become powerful, which is considered ''IT Industrialization.'' Researchers have also focused on functional operations [19]. Later, they added a control system to reduce the risks, termed ''Digitization.'' Finally, they made IT innovative, rapid, and flexible. Once this seemed perfect, there was rapid innovation in the so-called SDN. SDN has also been subjected to numerous trial-error methods. SDN 1.0 version bought up OpenFlow. As there were a few drawbacks from the first version, the SDN 2.0 version separated the overlays, and finally, the SDN 3.0 version was applied n-centric [20]. Following the development of IT and SDN. Then, a lot of changes in the field of technology were seen. In order to understand the evolution of programmable networks, we need to understand Figure 2. In this figure, it can be observed that there is a transition from the Traditional Architecture (left) to SDN Architecture (middle) and SDN Architecture with a Programmable Data Plane (right) using P4. Traditional architecture had tightly bound control and data plane, called ''tightly coupled.'' As SDN came into the picture, control, and data plane, are separated and had a bottom-up design. The information is shared using the OpenFlow Protocol. Currently, the new architecture seeks programmable data planes in SDN that followed a top-down approach. The programmable chip can be user-configured, whereas the SDN architecture has a fixed set of functions. SDN principles are well-defined and well-versed. All major cloud providers and data centers [21] use SDN. In the evolution of SDN virtualization, the king of virtualization called ''VMware'', acquired Nicira and renamed it as VMware NSX, which is the SDN-style network. NSX, which represents SDN, is used by hundreds of thousands of virtual machines in data centers worldwide [22].

TABLE 1. Evolution of IT and SDN.

Evolution	Functionalities
IT Craftsmanship	Functionally Operationally Focused
IT Industrialization	Control, Predictable and Low Risks
Digitalization	Speed, Innovation and Conventional
SDN 1.0	OpenFlow
SDN 2.0	Seperate Overlay
SDN 3.0	Application Centric Infrastructure

3) WHERE IS SDN NOW?

SDN is not dead; it is still alive and it works surprisingly well responding to what is going on in the current network market. Many researchers assert that SDN does not exist; however, this is incorrect. SDN is present in all virtualized networks and software-defined networks. Instead, it includes an SDN, allowing it to outperform its prior generations. This is similar to how analysts have predicted the future of SDN in the past. However, it is no longer referred to as SDN because it has been integrated into today's tools. As a result, it is referred to alongside its tool name, for example, SD-WAN (Software Defined Wide Area Networks).

4) WHERE SDN IS IMPORTANT?

SDN is significant and allows network operators to combine software and hardware from multiple vendors and create customized network services and infrastructure.

- Software-defined networking improves network connectivity for sales, customer service, internal communication, and document-sharing.
- SDN enables organizations to create customized network services and infrastructure by combining software and hardware from multiple vendors.
- A virtual network combines SDN, network function virtualization, and white boxes.
- SDN allows for faster deployment of new applications, services, and business models.

5) WHY DOES SDN NEED P4?

There are many challenges faced by SDN. Firstly, it requires time to implement new protocols and extend the functionality

of current protocols. Secondly, protocols like "OpenFlow" in SDN lack operational and administrative interfacing. And the most important issue is to add new protocols in the OpenFlow protocol, which is a research-driven time-consuming process. Therefore, in comparison to traditional fixed-function switches, P4 programmability opens up entirely new possibilities for network stack flexibility from 7-layered OSI or 5-layered TCP/IP modeling. Second, there is a need for upgradability (hardware solutions will never have), which allows adding functions and protocols to network devices by updating the P4 software running on them rather than purchasing new switches and their frameworks. In summary, SDN has become a promising architecture for the centralized management of network architectures making networks dynamic, centralized, flexible, and programmable. Therefore, it is necessary for the SDN to acquire P4 in its environment.

C. DATA PLANE PROGRAMMING

Data Plane Programmability is a buzzword owing to its varied scope and functionality [23]. P4 is a programming language that allows users to write custom protocols, build complex match/action pipelines, and include external functions in the code [24]. A programmable data plane is a flexible way to forward packets that can handle various formats and protocols [25]. Programmable data plane in the next step in enabling switches to perform complex packet operations. However, they cannot perform all tasks alone [26], [27]. P4 is used to configure the forwarding actions. P4 [28] was created for data-plane programming. As a result, P4 defines the actions that can be performed on packets [29], where the control and application plane are already programmable, and data plane programming is the missing link.

1) RISE OF P4

With the enormous growth in data traffic, new protocols, and private-public clouds, the rate of innovation in the networking world is increasing. The transition from traditional silicon switches to flexible programmable switches has now been completed and requires a new standard language. This new language must meet the following three basic requirements.

- The language should allow the network to be flexible.
- The language used should be expressive and comprehensive.
- The language must be portable from one architecture to the next.

The figure 3 P4 language's evolution and development are depicted. The P4 language draft was submitted in 2014, and it was first introduced globally in May 2015. The P4 community clarified the distinction between the OpenFlow and P4 languages in 2016, after many misunderstandings. In May 2016, the second version was introduced as $P4_{16}$, while the older version was known as $P4_{14}$.

The figure 3 depicts the evolution and development of the P4 language. The P4 language draft was submitted in 2014, and it was first introduced globally in May 2015. The P4 community clarified the distinction between the OpenFlow and P4 languages in 2016 after many misunderstandings. In May 2016, the second version was introduced as $P4_{16}$, while the older version was $P4_{14}$. P4Runtime was introduced and embedded into the control plane in July 2017 to run P4 programs. Many large-scale networks have adopted P4 due to scalability, feasibility, and reconfigurability. P4Runtime v1.0 was released in 2019, and P4Runtime v1.2 was released in 2020.

TABLE 2. P4 Versions.

$P4_{14}$	
Version 1.0.1	January 2015
Version 1.0.2	March 2015
Version 1.1.0	January 2016
Version 1.0.3	November 2016
Version 1.0.4	May 2017
Version 1.0.5	November 2018
$P4_{16}$	
Version 1.0.0	May 2017
Version 1.1.0	November 2018
Version 1.2.0	November 2018
Version 1.2.1	June 2020

As listed in Table 2, various versions of P4 have been developed since 2013. The initial idea began with a team of researchers in May 2013. Finally, the researchers submitted their first draft proposal for P4 to SIGCOMM in July 2014. Once they received approval, they transformed the paper and submitted it in August 2014. Finally, P4 is accepted as a programmable language. It was called $P4_{14}$. $P4_{14}$ was introduced in four different versions. However, $P4_{16}$ was introduced with new data types and control statements to fill loopholes in the $P4_{14}$. The programming language P4 is gaining popularity in the network industry [30].

P4 is the name of the first language release and $P4_{14}$ is the current language specification. $P4_{14}$ is a simpler language, but $P4_{16}$ has additional features that may not be compatible with $P4_{14}$ [31]. The current language $P4_{16}$ will be syntactically correct with the future versions of the language. Simultaneously, many native $P4_{14}$ language features have migrated into libraries of fundamental constructs required for writing effective P4 programs [32]. This includes target-specific implementations of certain functions (counters, meters, and checksum calculations), referred to as externs.

P4 (Programming Protocol-independent Packet Processors) is a domain-specific language used to control packets sent to data plane in networking devices. It consists of constructs, such as counters, registers, header field specifications, and support for matches and action tables. It is specially designed for the data plane, which checks the correctness of SDN switch behavior. With P4, custom header formats can be defined, and dynamic header parsing can be performed. The P4 program is written in a specific format and must include the following: header, parser, deparser, match + action, checksum verification, ingress, egress, and checksum computations. One section depends on the other in the program, which performs functions.

FIGURE 3. Evolution and history of P4 language.

2) WHY COMMUNITY DESIRES P4?

P4 is a network programming language that describes the packets processed using a programmable forwarding details. It is a programming language that allows network devices to be programmed with new features [33]. P4 provides a simple set of tools for imposing a community stack on the developer switching hardware. The tables with user-described keys with actions, counters, meters, and abstractions can be used as header types and partners. P4 has no understanding of how the Ethernet or IP headers look. Only the developer can tell the hardware of an Ethernet header, how to parse, match, and depart, and finally, which port the packet should be forwarded to. This programmability opens up new opportunities for flexibility of the network stack compared to traditional fixed-function. It can upgrade network devices with new features and protocols instead of buying a new switch.

D. RECENT TECHNOLOGY

Recent chip design schemes have demonstrated that flexibility can aachieve terabit-level speeds in custom ASICs [34]. Therefore, it is difficult to create a new generation of switching chips. Similar to microcode programming, each chip had its low-level interface. The switch was configured using P4. The forwarding table was sent to the fixed-function switch via an existing protocol interface (such as Open-Flow). P4 raises the abstraction level for the programming control network [35] and functions as a standard interface between the controller and the switch. In other words, researchers believe that the future OpenFlow protocol generation allows the controller to tell the switch how to do so without being constrained by the inherent design of the switch [36].

III. P4 ECOSYSTEM

This section provides an in-depth understanding of the P4. P4 is a domain-specific language used to program the data plane of the switch. It is target-independent and protocol-independent and allows the controller to define its packet analysis and header-processing process. While writing P4 program, three major goals must be met:

1) Target Independence: The P4 compiler should consider the switch's capabilities when converting the target-independent P4 description into a target-related program for configuring the switch.
2) Protocol independence: A switch doesn't need to be bound by a specific packet format. However, the controller should be able to specify the following:
   - A packet sniffer can extract header fields of specific names and types.
   - These header regions were processed using a collection of ''match-action'' tables.
3) Re-configurable: The controller should be able to refine the packet analysis process of the data packet and the processing process of the header area

A. P4 ARCHITECTURE

It is necessary to concentrate on understanding how P4 enters various functional processing blocks. P4 enters different blocks of the functionality and processes. Figure 4 and figure 5 describe how data packets are transmitted on various forwarding devices (such as Ethernet switches, load balancers, and routers). This has enabled the researchers to develop a universal language to describe how to process data packets by using the universal abstraction model. There are two different architectures: $P4_{14}$ and $P4_{16}$ architectures.

1) $P4_{14}$ ARCHITECTURE

Figure 4 depicts the $P4_{14}$ architecture. $P4_{14}$ architecture is called as ''P4 Abstract Forwarding Model''. $P4_{14}$ has targeted PISA-like devices and $P4_{16}$ has outgrown PISA. The PISA architecture in $P4_{14}$ is a protocol-independent switch architecture that is a single-pipeline forwarding architecture. It consists an input block, ingress match-action, buffers, egress match-action, deparser, and an output block connected to each other in a linear pipeline. The limitation of this architecture is: it does not know to express the packets which are processed in the pipeline. All the blocks are explained in detail in further sections.

2) $P4_{16}$ ARCHITECTURE

$P4_{16}$ uses a portable switch architecture, which allows it to target multiple programmable devices with different architectures. $P4_{16}$ uses a programmable parser and performs a multi-

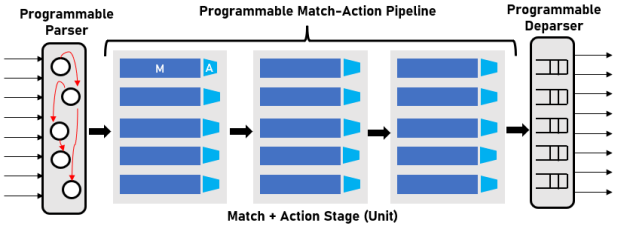

FIGURE 4. $P4_{14}$ Architecture.

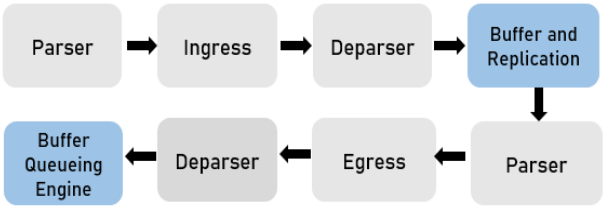

FIGURE 5. $P4_{16}$ Architecture.

stage matching-action process to forward incoming packets, which can be either serial or parallel. This architecture is widely used as it makes use of additional constraints to solve the problems.

Figure 5 depicts the $P4_{16}$ architecture. $P4_{16}$ follows a Portable Switch Architecture and can be implemented on any switch target. The multiple networking functions implemented in the PSA are combined into a single pipeline. The pipeline has the following components: parser, checksum validate, ingress match-action, packet buffer, egress matching action, checksum update, deparser, and buffer queuing engine.

The switch uses a programmable parser and performs a multi-stage matching-action process to forward incoming packets, which can be serial or parallel. The ABS model [37] contains three generalizations based on OpenFlow research. OpenFlow assumes that the ''matching-performing actions'' stages are sequential, and the model allows for parallel/serial execution. The model assumes that ''actions'' are written using switch-supported protocol-independent primitives.

B. P4 COMPONENTS

The P4 program contains definitions of the following key elements:

1) Headers: The sequence and structure of the series of header areas are described in the header-definition section. Each header is processed by a state machine that uses the value of the current header to trigger the next transition in the state machine. This specifies the length of the area and limits the value of data in the area. The header-type P4 was similar to the C structure. In addition to the standard packet headers, P4 allows custom headers to be processed, including Ethernet, IP, TCP, and UDP. The standard and custom headers were explicitly processed. For example, standard ethernet and ipv4 headers in P4 are specified as below.

```
/* HEADERS */
struct metadata {...}
struct headers {
   ethernet_t  ethernet;
   ipv4_t      ipv4;
}
```

2) Parser: A parser's definition explains how to recognize the header and the effective header sequence in a data packet. The parser in the program specifies how packet headers are parsed. It checks and identifies the header present in the incoming packets. After parsing, the parsed packets were sent to the control block. The underlying switch can implement a state machine that traverses each header of the data packet from start to finish, thereby extracting the value of the header area while it is running. The extracted header area value is sent to the ''match-action'' table for processing. P4 describes the state machine as a collection of transitions from one header to another. The current header value can then be used to trigger the transition. For example, the standard structure of the parser is specified below.

```
/* PARSER */
parser MyParser(packet_in packet,
        out headers hdr,
        inout metadata meta,
        inout standard_metadata_t  smeta, {
    ...
}
```

3) Table: The mechanism for packet processing is the ''Match-Action'' table. The P4 program's header area can be used for matching or performing specific actions on it. The programmer then describes how the defined header fields should be matched in the match+action stages, as well as what actions should be taken when a match occurs. The programmer creates a table based on these fields and then performs an action to add the mTag header. For example, the standard structure of the table is specified below.

```
/* TABLE */
table source_check {
        reads {
            mtag : valid;
            metadata.ingress_port : exact;
}
/* ACTIONS*/
actions {
      fault_to_cpu;
      strip_mtag;
      pass;
}
 max_size : 64;
}
```

4) Actions: P4 includes a set of actions from which more complex actions can be constructed. Every P4 program declares a set of action functions made up of action primitives; these action functions make table

specification and population easier. P4 allows complex execution actions to be created using simple, protocol-independent primitives. The ''Match-Action'' table can be used to implement these complex actions. For example, the standard structure of the actions is specified above within the table structure.

5) Deparser: This serializes the modified headers back into the packet as the packet moves from one switch to another. The function considers the packet-out variable and header values as two arguments in the pipeline. For example, the standard structure of the deparser is specified below.

```
/* DEPARSER */
control MyDeparser (inout headers hdr,
                    inout metadata meta) {
    ...
}
```

6) Control Program: Defining the flow of control from one table to the next is the only task left after defining the tables and actions. Through a combination of functions, conditionals, and table references, control flow is specified as a program. The control program determines the sequence of processing data packets in the ''Match-Action'' table. A simple and necessary program describes the control flow between the ''match-action'' tables. These control blocks were used to perform matching and action tables for packets based on different header fields. The imperative representation of this packet processing pipeline is as follows:

```
/* CONTROL PROGRAM */
control main() {
    table(source_check);
    if (!defined(metadata.ingress_error)) {
        table(local_switching);
        if (!defined(metadata.egress_spec)) {
            table(mTag_table);
        }
        table(egress_check);
    }
}
```

C. P4 WORKFLOW

This section discusses the working flow of P4 in detail. Figure 6 shows the workflow for P4. The P4 program comprises two operations: configuration and delivery. The configuration operation determines the order of each stage of the ''matching-perform action'' and specifies the protocol header area to be processed in each stage. By configuring the switch, we can determine which networks are supported and how the data packets can be handled. The operation issued adds or removes the entry from the ''Match-Action'' table. The table itself is one of these, and it is specified during the configuration operation. The issuing process determines the execution strategy used by the data packet at any given time.

The working flow of the P4 program is depicted in Figure 6. Each program block performs a unique function. Each section of the P4 program is linked to a different block that performs a specific function. Furthermore, as shown in 6, the P4 program is divided into four major sections: data

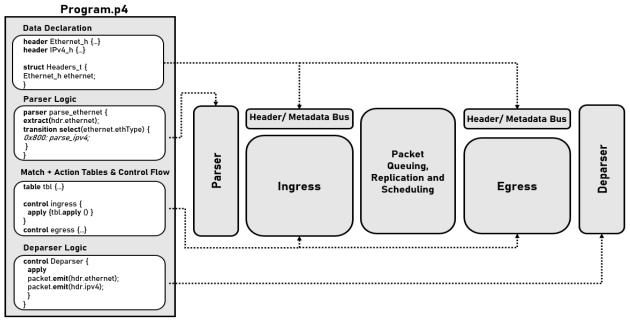

FIGURE 6. Working Flow of P4 Program.

declaration, parser logic, a match-action table with control flow, and deparser logic. A parser, ingress, scheduler, egress, and deparser are required to run the P4 program successfully.

The packet parser is the first to process the arriving data packets and search for specific areas from the packet header, specifying switch protocol support. This model makes no assumptions about the meaning of the protocol header. Instead, the parsed data packet representation defines a set of header regions on which the 'matching-performing actions' process is performed.

Data packets contain metadata, which is extra information that can be passed between processing stages. Metadata can be used as a packet header area as well. Examples of metadata include ingress port numbers, transmission destinations and queues, timestamps used for packet scheduling, and data passed between the tables. Data such as the virtual net identification number did not change the performance of the data packets after parsing.

The match-action table received the extracted header area. There are two sections in the match action table: the entry and exit tables. The entry table determines the queue in which a data packet is placed. Data packets can be forwarded, copied, discarded, or triggered by flow-control-based ingress processing. When a packet is copied by multi-cast, the egress ''match-action'' is modified separately for each action target on the packet header. An action table is linked to tracking the status of each frame.

IV. DATA PLANE PROGRAMMABILITY

Data-plane programmability refers to the ability of a network device to expose low-level packet-processing logic through a standardized API, facilitating comprehensive and methodical reconfiguration. The creation and acceptance of the SDN paradigm have greatly expanded the flexibility, and dynamicity of device functionality [38]. Conventional network equipment cannot often be updated throughout its lifetime because the data plane functionality is tightly integrated into the hardware and software of the device. In software-based packet-processing systems, considerable vendor-software changes are required to alter the functionality of the data plane. This survey is broadly classified into six sections: SDN with data-plane programmability, data-plane architectures, data-plane

abstractions, data-plane network monitoring with data-plane Programmability, P4 challenges, and data-plane applications. Tables 3, 4, and 5 present a detailed review of the existing studies. The primary contributions of this study are as follows.

- This study analyzes and highlights the key factors that have evolved from traditional network architectures to P4-Programmable SDN architectures.
- Detailed understanding of P4 components, their architecture, and workflo is explained in-detail
- Literature review is done on the following parameters: performance, security, load balancing, along with its strengths and weaknesses. It which gives overall novel summary for P4-based solutions.
- A major problem in the data plane programming area is identified and discussed in the further section.

A. SDN WITH DATA-PLANE PROGRAMMABILITY

According to the survey, softwarization and current SDN platforms may not be able to handle the complexity and heterogeneity of many needs, such as strict latency, jitter, high-accuracy traffic, and advanced monitoring. SDN/NFV [39] must be upgraded to enable these services, considering data plane programmability in addition to orchestration and control plane [40]. A software layer for softwarization was created to overcome these limitations, allowing the switch data plane to be programmed using high-level programming languages and APIs. Thus, the P4 language curriculum is becoming increasingly essential and well-planned. With its cutting-edge SDN/NFV softwarization, P4 initially developed for intra-data census scenario, has rapidly gained prominence. The difficulties encountered by P4 are discussed, along with the solutions, in Table 3.

B. DATA-PLANE ARCHITECTURES

A survey of the data plane architecture showed that low-level programmability is now supported by a larger range of devices and operations, despite the fact that switches are initially the primary focus of data plane programmability (particularly in data centers) [41]. Programmable data plane hardware or software is increasingly used for middleboxes (such as firewalls or load balancers) and general network processing, in addition to packet switching. Additionally, programmable NICs at the network edge enable data plane programmability [42]. These gadgets can be realized on top of one or more architectural designs. However, the difference between the software and hardware data plane was hazy. For instance, a hardware-based device might nevertheless use a general-purpose CPU to execute activities in which the underlying hardware is not designed to handle or that do not require significant performance. Current software switches increasingly rely on domain-specific hardware features, such as SmartNIC, which offloads packet processing logic to hardware via Data Direct I/O (DDIO) and Receive Side Scaling (RSS) [43].

C. DATA PLANE ABSTRACTIONS

Control plane technologies differ in a number of aspects, such as the primitives for packet processing they offer and the language constructs that can be used to combine these incentives to build specific pipelines [44]. The following sections examine some of the most common abstractions used and disclosed in programmable data plane systems in light of this inherent architectural relationship. The first step involves discussing programmable packet-processing pipelines before moving on to abstraction for packet parsing and scheduling. The final step in our exploration of programming languages and compilers is their examination.

1) PROGRAMMABLE PACKET PROCESSING PIPELINES

The main advantage of programmable data plane is the adaptable packet processing. The two main abstractions on top of which modern packet processing pipelines are frequently built are the match-action pipeline abstraction and dataflow graph abstraction. To create programmable switches, early packet processing systems adopted data flow graph abstraction [45] and machine learning [46]. In addition, stream processing frameworks, such as Apache Flink and Spark, heavily utilize this architecture. When representing the processing logic as a graph, a data flow graph uses nodes to represent the basic computing steps and edges to indicate the data movement between these stages.

2) MATCH-ACTION PROCESING

Data-plane applications are described using a hierarchical set of lookup tables (flow tables) in match-action abstraction [47]. Proposed a model for the effects of match-action tables on the routers. For packets to be processed correctly, a table query is performed based on portions of the header fields. This allows the switch to perform actions, such as rewriting packet components, encapsulating or decapsulating tunnel headers, dropping packet, forwarding packets, and packet processing delays until flow tables are created [48].

3) PROGRAMMING LANGUAGES AND COMPILERS

SDN initiatives, such as OpenFlow, ForCES, and NETCONF, have taken the field beyond low-level protocols over the past few years. The declaration of packet processing policies within a specific switch architecture is made possible by new high-level data plane programming languages in terms of abstract, general, and modular language constructs [49]. Operator expectations for more complex SDN applications are the main driving forces behind these initiatives. Owing to the availability of more programmable line-rate networking hardware, linguistic methods have been used to specify line-rate switching architectures (i.e., match-action tables and protocols supported by the parser) [44].

TABLE 3. SDN with Data-plane Programmability.

Reference	Objective	Results	Techniques used	Strengths	Weakness
[2]	Comparison with traditional V/s SDN.	1. Network Issue Troubleshooting 2. Scalability 3. Reliability 4. Performance and 5. Security	OMNET++, Mininet, NS-3 and Estinet	Differentiate Control Plane into three primary layers.	There is no clear picture of how communication occurs between control and data plane.
[3]	A detailed survey was conducted using OpenFlow standards.	Researchers have implemented this simulation for oil-company networks.	Ryu and Mininet Simulation using Python Scripts.	Discusses the features of abstraction and SDN layers.	Lacks of exact information regarding increasing network infrastructure demands.
[5]	SDN is considered a key solution to meet the increasing demand for networking infrastructure.	This results in the implementing of diverse configurations and keeping track of numerous events in the network.	Continuous research is being carried out on the SDN model.	Data centers, enterprise networks, small enterprises, and home networks have all adopted SDN.	The SDN paradigm is still being explored and developed across various network-management platforms.
[6]	Focusing on the data plane programming model.	P4Runtime results as the control-to-data plane protocol.	P4 and P4Runtime used as a tool for fog computing.	The data plane model serves as the first step in evaluating next-generation SDN data plane	Full Programmability of Next-Generation SDN Networks.
[12]	Recognizing innovation and cutting-edge SDN methods and concepts for 5G networks	They were brought together to discuss technical challenges and recent advances in 5G SDN.	A Survey on 5G cellular network architecture and on some of the emerging technologies.	Developing advanced and innovative SDN techniques for 5G networks for academia and industry.	5G-SDN faces many challenges and technical issues before it is widely deployed.

TABLE 4. Network monitoring with Data-plane Programmability.

Reference	Objective	Results	Techniques used	Strengths	Weakness
[13]	Major challenges in Data-plane connection diagnosis involves the identification of a "sweet spot" for diagnosis.	Analyze TCP performance in real-time near the end hosts.	Make use of P4 to prototype Dapper and evaluate the design on synthetic traffic.	Author introduces a system called "Dapper" which analyzes the TCP performance.	Design challenges and the necessary steps for Dapper to run in the data plane.
[18]	Evaluates compilers or target combinations and helps a larger community for tool optimization.	Black box approach for collecting measurements of sufficient granularity in performance.	P4 compilers: PISCES, P4FPGA, BMV2, Xilinx prototype based on SDNet.	Whippersnapper is a synthetic benchmark suite, which systematically evaluates the key language components.	To identify the results with interesting opportunities for optimization of the system.
[27]	Data Plane and Control Plane are given more attention and act as critical pieces of the puzzle.	This results in deciding the optimal number of CPU cores for the throughput requirement.	L2 and L3 Forwarding, Data Center Gateway, Scalability, and Portability.	To investigate the scalability of increasing the number of cores based on run-time core reallocation.	Data Plane has received very less attention compared to the Application and Control Plane.
[34]	Presents the design and implementation of a Priority-based Adaptive Multicast, and discusses the analysis and evaluation	PAM optimizes intra-datacenter multicast transfers in clouds for single tenants.	1. Multicast Routing Technique 2. Multicast in Layer 7	Provides distributed mechanism in scheduling to approximate a range.	The working of PAM in real environment is still unclear.
[36]	To configure switches and process custom packet headers using match and action tables.	The approximations are accurate, and they do not exceed hardware resources.	Protocols for congestion control and load balancing, such as XCP, RCP, and CONGA	Address the limitations by providing a set of generic building block techniques.	To operate at line rate, and limit per-packet computation.
[52]	Focuses software-based data processing which is typically slow.	Presents a P4 language-based generator of high-speed input (Parser) and output (Deparser.)	Xilinx Virtex-7 XCVH580T FPGA	The tool converts a P4 description into synthesizable VHDL code that can be used on an FPGA.	The development of new Parser and Deparser architectures that enable us to scale throughputs.

D. NETWORK MONITORING WITH DATA-PLANE PROGRAMMABILITY

The most exciting data-plane offloading applications are network metering, measurement, monitoring, and de-bugging. This is primarily due to the fact that these applications share certain characteristics that make them especially well-suited for data plane-based implementations: They must meet strict performance standards and handle a tremendous amount of

TABLE 5. The current P4 Challenges in the programmable data-planes.

Reference	Objective	Results	Techniques used	Strengths	Weakness
[4]	A dedicated comprehensive survey was conducted on SDN and future challenges.	Verify forwarding tables in very large networks	STRIDE Methodology, SDN Compass, Panopticon, and Security Methodology.	Proposes a layered approach to dissect the state of the art in terms of the concepts, ideas, and SDN components.	In large-scale networks, not possible to assume that the network is consistent, because of frequent changes.
[11]	SDN, NV, NFV, APIs, data plane, middleboxes, SDN security, hybrid networks, IoT, and 5G are surveyed.	Results in network operators advancing and broadening adoption.	An extensive survey was conducted on various SDN, Cloud, IoT, and 5G Networks streams.	The following factors were measured: latency, jitter, flow rate, redundancy, security, and cost.	Aims to identify where new and existing researchers can still contribute to SDN advancement.
[23]	The goal was to concentrate on data-plane programming models and the P4 programming language.	Results of development and deployment optimization, as well as P4-specific approaches to control plane operation	Flow Monitoring, and Traffic Aggregation, are some of the services offered.	Research initiatives to advance P4 technology, and tutorial on data plane programming models	Categorizing and summarising a large body of research domains.
[35]	A taxonomy of applications written in the P4 language that is unique and comprehensive.	The challenges and considerations present future perspectives and open research issues.	INT, MiddleBox functions, network performance, IoT, cybersecurity and attacks, network, and P4 Testing.	Memory capacity, network security, modularity, interoperability, and control plane intervention.	The traditional control plane and transition to SDN are discussed to demonstrate the evolution of networking.

traffic. The data plane can directly and quickly access the input data used by the monitoring program, such as, the network packets or data plane measurements. The performance of P4, when combined with modern networking tools, is presented in Table 4. Furthermore, we examine P4 performance in more detail and show how P4 measures up in terms of TCP, Black Box approach, CPU cores, and PAM, among other metrics.

In [50], a number of solutions were presented to improve insight into network behavior. One of these solutions is the Tiny Packet Program (TPP) interface, which allows end hosts to actively access and modify internal network data. It is based on the Active Network concept and works on Smart Packets [51], which were first presented for switch network administration and monitoring. The ''division of labor'' theory forms the basis of the strategy: End hosts perform flexible computation on the network state switches move forward while carrying out TPPs in-band at line rate.

E. P4 CHALLENGES

Both the science and art of designing programmable switches are based on abstraction. An abstraction should ideally be basic enough to encapsulate the optimal amount of changeable data plane functionality, while also being expensive enough to enable the top layers to synthesize complex packet processing behavior. Additionally, the control plane should be able to use such an abstraction with ease, owing to a reliable and secure data plane API [53]. It must provide a well-defined consistency mechanism and effectively handle the global states in the data plane. It should also allow automatic program alterations to improve performance [54] and analytical performance models. It must distinguish between dynamic behavior and static semantics. Additionally, a useful conceptual framework should be included. The survey demonstrated that P4 worked well, and could be applied to extensive networking. The OpenFlow Protocol and P4 Language explanations are provided in Table 5. Both are unique and fulfill functions that are unique to them. discusses the P4 programming language and SDN. The P4 language and the OpenFlow Protocol are both confusing. The majority believe that P4: eventually replaced OpenFlow, and P4 and OpenFlow became interchangeable. However, this was not the case. Both are distinct from each other and excel in their role.

F. DATA PLANE APPLICATIONS

With the introduction of programmable data plane, network devices can be configured to perform specific generic information-processing capabilities, such as telemetry, massive data processing, machine learning, and even whole key-value stores. The ability to configure network devices allows a dumb pipe that can only move data to suddenly become a complete and, a sophisticated data-processing pipeline that can transform data as it passes. Load balancing can be achieved by connecting resilient routing to programmable data plane, which can then be used to distribute traffic dynamically across several forwarding channels, workers, and back-end servers. For example, Hedera [55] compared their results to those of a load balancer. The ''resource pooling'' idea is to be realized via horizontal scaling, which makes a group of different resources behave like a single pooled resource in order to benefit from statistical multiplexing, load dispersion, and improved failure resilience. HULA [56] is a scalable load-balancing approach that uses programmable data planes to overcome the shortcomings of ECMP routing and currently employs congestion-aware load-balancing techniques, such as CONGA [57]. HULA is adaptive and scalable because each switch tracks only the congestion for the shortest path to a destination through an adjacent switch. Another example of

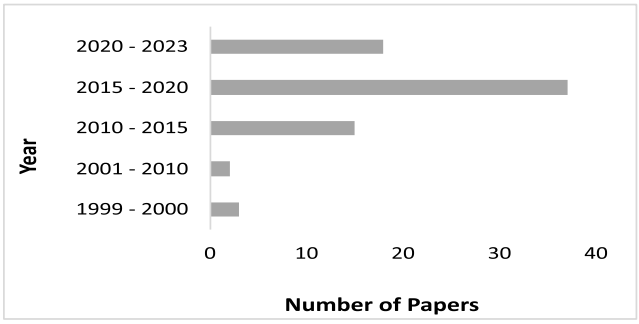

FIGURE 7. Distribution of surveyed research works per year.

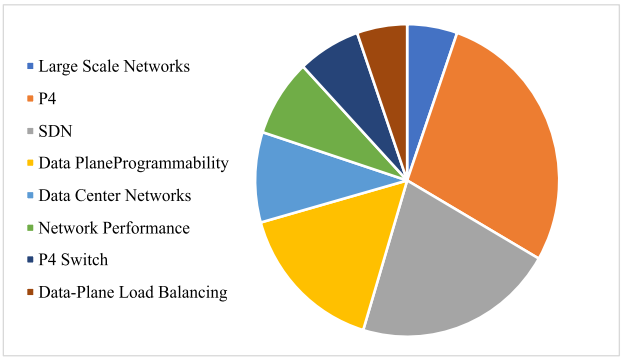

FIGURE 8. Distribution of studied papers based on different categories.

a load-balancing application is SilkRoad [58], which employs programmable ASICs to produce faster load balancers.

V. RESEARCH GAPS AND CHALLENGES

The authors noted significant research voids in P4- programmable switches, which they intended to fill by demonstrating that P4-based solutions for DDoS detection, load balancing, packet aggregation, and disaggregation are the components of community monitoring. The paper distribution of the surveyed research papers is depicted in Figure 7 from 1999 to 2023. The 75 papers are currently being reviewed.

Figure 8 depicts the distribution of studied papers across various categories. SDN, P4, Large-Scale Networks, Data Plane Programming, P4 Switch, Data-Plane Load Balancing, Data Center Networks, and Network Performance are the broad categories of the papers.

A. RESEARCH GAPS

From the reviewed papers, the observed research gaps are as follows:

- Most researchers test their operation on non-functional BMv2 switches because they do not support configuration, monitoring, and operating protocols, such as OpenConfig, gNMI, and gNOI. Furthermore, compared with physical hardware switches, the computing power of the BMv2 switches is quite low. The authors noted that there are significant research voids in P4- programmable switches and that production-ready platforms such as stratum-based switches will allow the research community to test P4-based solutions.
- The currently available P4-based solution does not distinguish flash traffic from attack traffic, and attackers can use low-frequency DDoS attacks to circumvent security solutions.
- Some DDoS defense solutions use synthetic data sets generated by local traffic generation tools. There are no reference data sets that include both malicious and normal traffic. Some research papers use small topologies in their experiments, making validation impractical.
- For switch ASICs, some authors implemented load-balancing algorithms using P4. However, data connections must be stored in a large amount of memory within a switch. Another research area is the development of efficient load-balancing solutions for data plane switches with limited memory.
- Most network-monitoring solutions require a significant amount of computational resources, which may cause performance bottlenecks. Only a few studies have divided monitoring tasks among data plane switches. Although these solutions improve network performance, they have synchronization and communication overheads. A major research challenge is to perform distributed monitoring in programmable data planes with a minimum amount of synchronization and communication overhead.
- The implementation of network-monitoring solutions is expensive. Research is required to identify the root cause of network problems, but none of these solutions uses popular open-source SDN controllers. Therefore, flow statistics are reported only in the event of a network issue. However, it is possible to result in the omission of some information that is required to determine the source of the problem. Debugging network problems and identifying root causes.
- Ryu, OpenDaylight, or ONOS are used to control the P4 programmable data plane. Therefore, a major research topic is integrating P4 programmable data planes with SDN controllers because it facilitates the migration of existing SDN applications to the SDN-programmable data plane model and enables the centralized management of P4 programmable data planes.
- Despite reducing the control plane computation and communication overhead, P4-based solutions can increase or reduce switch complexity in the data plane. Research is still needed to determine how to deploy P4-based algorithms in switches while minimizing complexity effectively.
- Currently, research is focusing on programmable switches. However, the real challenge in fully deploying these solutions is replacing all the existing switches with programmable switches with fixed functions. It is an unresolved research problem to integrate fixed-function

switches alongside programmable switches and check the performance of the network.

B. CHALLENGES

In this section, we discuss the research challenges based on state of art literature.

1) DESIGN ISSUES IN P4

P4 did not support loop constructs, pointers, references, or dynamic memory allocations. Consequently, algorithms that require deep packet inspection are difficult to implement [59]. Because P4 does not support arithmetic logarithmic functions, the DDoS detection algorithms were implemented by estimating the entropy value rather than computing the exact entropy using the longest prefix match tables. This is an open research challenge, owing to the inclusion of linguistic constructs. As a result, the research community is confronted with an open research question.

2) COSTS AND P4-PROGRAMMABLE SWITCH AVAILABILITY

Only a few firms produce P4 ASICs [60], and replacing a fixed-function device with a programmable device is more expensive than adding a traditional device to an existing network [61]. P4 specialists are also required to define the behavior of the programmable switches [62], and their integration of SDN and traditional networks remains a research challenge. To define the behavior of programmable switches, P4 specialists are also needed. The integration of SDN with traditional networks [63] remains a research challenge. SDN networks have also become more complex owing to the addition of programmable data planes. To solve these problems, incremental deployment is necessary. This requires gradually introducing programmable and fixed-function SDN devices into traditional networks as viable options. The Paxos protocol [64] is the foundation for several fault-tolerant distributed systems and services. The implementation of Paxos is a critical use case for P4 and will help shape the data plane language requirements in general. PISCES [65] is a software switch derived from OpenvSwitch (OVS), a hard-wired hypervisor switch whose behavior can be customized using P4. PISCES is not hardwired to any specific protocol, making it simple to add new features.

3) DATA PLANE SECURITY

Active networks [66] was the first research effort on programmable networks in the late 1990s. Therefore, the concept of programmable networks is a new one. However, it has not been widely adopted owing to its lack of security. Similarly, the P4-Programmable data plane has some security issues. Bugs are used more commonly in software applications than in hardware applications. In addition, the forwarding behavior of the [67] data plane is usually determined by less experienced and less prudent end users than by providers. Second, attackers can use programmability to change the forwarding behavior of the device to mitigate new attack vectors. Therefore, it is necessary to add assertions to programs and perform verifications to improve the security of the programmable data planes. To protect network links between P4-based SDN switches, MACsec [68], a widely used IEEE standard for securing Layer-2 infrastructures, will be deployed automatically. The MACsec is supported by a wide range of switch and router manufacturers. It has only minor performance limitations on these devices compared to VPN technologies such as IPsec. P4-MACsec is a MACsec data-plane implementation proposal.

4) ABSTRACTIONS

Both the science and art of designing programmable switches are based on abstraction. In a perfect world, an abstraction would be simple enough to encompass just the right amount of configurable data plane capability while simultaneously being expressive enough to allow the top layers to synthesize complicated packet processing behavior. Additionally, the control plane should be able to easily use such an abstraction, owing to a reliable and secure data-plane API [69], [70].

5) RECONFIGURABILITY

The goal of moving from OpenFlow to P4 is to expose a switch's processing capabilities flexibly and efficiently, including programmable packet parsing and general scheduling and queuing systems [71]. It extends beyond how packet-processing regulations are modeled in the data plane, including how packets are linked to specific processing operations. In particular, changing the data-plane behavior at runtime is difficult without affecting packet processing [72].

6) SCALABILITY

Designers are investigating more complex solutions to handle the application states in the data plane [73]. However, stateful techniques for packet processing are still in their infancy, and no clear leader has yet been identified, whereas stateless approaches are currently steady. Stateful abstraction is challenging because it must address state management challenges (such as consistency) in a manner that is user-friendly for programmers and that guarantees high performance. This is particularly difficult because one of the leading causes of performance problems in contemporary computer systems is still frequent reading and writing to memory, which is constantly required in workloads involving packet processing [74].

7) DATA PLANE VULNERABILITIES

The data plane composition, or downward mapping of the data plane from the intent layer, is a part of this issue. The difficulty of automatically adjusting the network to changing environments is closely tied to the need to verify the accuracy and intended impact of the configuration adjustment. The control loop must include an upward mapping of the data

plane from the intent layer, and the data plane must be operated securely, because programming increases the possibility of introducing vulnerabilities.

Recent discoveries suggest that the network should be built with security and verifiability in mind from the very beginning [75], demanding new abstractions that must be operated securely because programmability increases the possibility of introducing vulnerabilities and additional attack surfaces. This is performed to verify the accuracy of the proposed method.

VI. CONCLUSION AND FUTURE TRENDS

This article provides a survey of programmable data plane and the related taxonomy is discussed. The P4 language is briefly described, along with research gaps and future trends. This survey examines the transition from traditional, to SDN to programmable SDN using P4. Following this, the evolution of networking, the importance of data-plane programmability, and the abstract model structure of P4 are discussed in detail. P4, the de-facto language used for programming the data plane of SDN is briefly described. The survey sheds light on numerous significant works and compares schemes within each category in the taxonomy, as well as with the legacy approaches. In the research challenges, we discuss various future trends and initiatives. This research leads us to the conclusion that open-source architectures and data network programming will dominate the network domain in the future. Thus, SDN + P4 would be an ideal environment in the upcoming years.

ACKNOWLEDGMENT

There is no conflict of interest among the authors and the publication body. We acknowledge the support of Prof. Raja Jurdak and Queensland University of Technology for this research.

REFERENCES

[1] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[2] S. H. Haji, S. R. Zeebaree, R. H. Saeed, S. Y. Ameen, H. M. Shukur, N. Omar, M. A. Sadeeq, Z. S. Ageed, I. M. Ibrahim, and H. M. Yasin, "Comparison of software defined networking with traditional networking," *Asian J. Res. Comput. Sci.*, vol. 9, no. 2, pp. 1–18, 2021.

[3] H. A. Eissa, K. A. Bozed, and H. Younis, "Software defined networking," in *Proc. 19th Int. Conf. Sci. Techn. Autom. Control Comput. Eng. (STA)*, Mar. 2019, pp. 620–625.

[4] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[5] A. O. Jefia, S. I. Popoola, and A. A. Atayero, "Software-defined networking: Current trends, challenges, and future directions," in *Proc. Int. Conf. Ind. Eng. Oper. Manag.*, 2018, pp. 1677–1685.

[6] E. O. Zaballa, D. Franco, M. Aguado, and M. S. Berger, "Next-generation SDN and fog computing: A new paradigm for SDN-based edge computing," in *Proc. 2nd Workshop Fog Comput. IoT (Fog-IoT)*. Wadern, Germany: Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020, pp. 9:1–9:8.

[7] N. Anerousis, P. Chemouil, A. A. Lazar, N. Mihai, and S. B. Weinstein, "The origin and evolution of open programmable networks and SDN," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 3, pp. 1956–1971, 3rd Quart., 2021.

[8] A.-A. Agape, M. C. Danceanu, R. R. Hansen, and S. Schmid, "P4Fuzz: Compiler fuzzer for dependable programmable dataplanes," in *Proc. 22nd Int. Conf. Distrib. Comput. Netw.*, Jan. 2021, pp. 16–25.

[9] D. Barach, L. Linguaglossa, D. Marion, P. Pfister, S. Pontarelli, and D. Rossi, "High-speed software data plane via vectorized packet processing," *IEEE Commun. Mag.*, vol. 56, no. 12, pp. 97–103, Dec. 2018.

[10] G. Antichi, T. Benson, N. Foster, F. Ramos, and J. Sherry, "Programmable network data planes (Dagstuhl seminar 19141)," in *Dagstuhl Rep.*, vol. 9, no. 3, pp. 178–201, 2019.

[11] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, and H. L. Owen, "Advancing software-defined networks: A survey," *IEEE Access*, vol. 5, pp. 25487–25526, 2017.

[12] M. Peng, I. Chih-Lin, C. Tan, and C. Huang, "IEEE ACCESS special section editorial: Recent advances in cloud radio access networks," *IEEE Access*, vol. 2, pp. 1683–1685, 2014.

[13] M. Ghasemi, T. Benson, and J. Rexford, "Dapper: Data plane performance diagnosis of TCP," in *Proc. Symp. SDN Res.*, Apr. 2017, pp. 61–74.

[14] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Comput. Netw.*, vol. 72, pp. 74–98, Oct. 2014.

[15] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: From concept to implementation," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 2181–2206, 4th Quart., 2014.

[16] P. Zanna, P. Radcliffe, and K. G. Chavez, "A method for comparing OpenFlow and P4," in *Proc. 29th Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, Nov. 2019, pp. 1–3.

[17] W. Braun and M. Menth, "Software-defined networking using OpenFlow: Protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, May 2014.

[18] H. T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soulé, and H. Weatherspoon, "Whippersnapper: A p4 language benchmark suite," in *Proc. Symp. SDN Res.*, Apr. 2017, pp. 95–101.

[19] D. F. Macedo, D. Guedes, L. F. M. Vieira, M. A. M. Vieira, and M. Nogueira, "Programmable networks—From software-defined radio to software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 2, pp. 1102–1125, 2nd Quart., 2015.

[20] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: An intellectual history of programmable networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[21] A. L. V. Caraguay, L. I. B. Lopez, and L. J. G. Villalba, "Evolution and challenges of software defined networking," in *Proc. IEEE SDN Future Netw. Services (SDN4FNS)*, Nov. 2013, pp. 1–7.

[22] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, and P. Dely, "Towards QoE-driven multimedia service negotiation and path optimization with software defined networking," in *Proc. 20th Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM)*, Sep. 2012, pp. 1–5.

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

[24] E. Kaljic, A. Maric, P. Njemcevic, and M. Hadzialic, "A survey on data plane flexibility and programmability in software-defined networking," *IEEE Access*, vol. 7, pp. 47804–47840, 2019.

[25] H. Mostafaei, M. Shojafar, and M. Conti, "TEL: Low-latency failover traffic engineering in data plane," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 4, pp. 4697–4710, Dec. 2021.

[26] W. L. da Costa Cordeiro, J. A. Marques, and L. P. Gaspary, "Data plane programmability beyond OpenFlow: Opportunities and challenges for network and service operations and management," *J. Netw. Syst. Manage.*, vol. 25, no. 4, pp. 784–818, Oct. 2017.

[27] P. G. K. Patra, F. E. R. Cesen, J. S. Mejia, D. L. Feferman, L. Csikor, C. E. Rothenberg, and G. Pongracz, "Toward a sweet spot of data plane programmability, portability, and performance: On the scalability of multi-architecture P4 pipelines," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2603–2611, Dec. 2018.

[28] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[29] H. Wang, R. Soulé, H. T. Dang, K. S. Lee, V. Shrivastav, N. Foster, and H. Weatherspoon, "P4FPGA: A rapid prototyping framework for P4," in *Proc. Symp. SDN Res.*, Apr. 2017, pp. 122–135.

[30] S. Kaur, K. Kumar, and N. Aggarwal, "A review on P4-programmable data planes: Architecture, research efforts, and future directions," *Comput. Commun.*, vol. 170, pp. 109–129, Mar. 2021.

[31] A. Liatifis, P. Sarigiannidis, V. Argyriou, and T. Lagkas, "Advancing SDN from OpenFlow to P4: A survey," *ACM Comput. Surv.*, vol. 55, no. 9, pp. 1–37, Sep. 2023.

[32] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computation is a dumb idea whose time has come," in *Proc. 16th ACM Workshop Hot Topics Netw.*, Nov. 2017, pp. 150–156.

[33] W. Braun, J. Hartmann, and M. Menth, "Demo: Scalable and reliable software-defined multicast with BIER and P4," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 905–906.

[34] S. Luo, H. Yu, K. Li, and H. Xing, "Efficient file dissemination in data center networks with priority-based adaptive multicast," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1161–1175, Jun. 2020.

[35] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE Access*, vol. 9, pp. 87094–87155, 2021.

[36] N. K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter, "Evaluating the power of flexible packet processing for network resource allocation," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 67–82.

[37] L. S. Peter, H. Kobo, and V. M. Srivastava, "A comparative review analysis of OpenFlow and P4 protocols based on software defined networks," in *Proc. ICDICI*, 2022, pp. 699–711.

[38] O. Michel, R. Bifulco, G. Retvari, and S. Schmid, "The programmable data plane: Abstractions, architectures, algorithms, and applications," *ACM Comput. Surv.*, vol. 54, no. 4, pp. 1–36, 2021.

[39] G. P. Katsikas, T. Barbette, D. Kostic, R. Steinert, and G. Q. Maguire Jr., "Metron: NFV service chains at the true speed of the underlying hardware," in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2018, pp. 171–186.

[40] F. Paolucci, F. Cugini, P. Castoldi, and T. Osinski, "Enhancing 5G SDN/NFV edge with P4 data plane programmability," *IEEE Netw.*, vol. 35, no. 3, pp. 154–160, May 2021.

[41] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful SDN data planes," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1701–1725, 3rd Quart., 2017.

[42] L. Linguaglossa, S. Lange, S. Pontarelli, G. Rétvári, D. Rossi, T. Zinner, R. Bifulco, M. Jarschel, and G. Bianchi, "Survey of performance acceleration techniques for network function virtualization," *Proc. IEEE*, vol. 107, no. 4, pp. 746–764, Apr. 2019.

[43] T. Lévai, G. Pongrácz, P. Megyesi, P. Vörös, S. Laki, F. Németh, and G. Rétvári, "The price for programmability in the software data plane: The vendor perspective," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2621–2630, Dec. 2018.

[44] T. Barbette, C. Soldani, and L. Mathy, "Fast userspace packet processing," in *Proc. ACM/IEEE Symp. Architectures Netw. Commun. Syst. (ANCS)*, May 2015, pp. 5–16.

[45] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured design," *IBM Syst. J.*, vol. 38, nos. 2–3, pp. 231–256, 1999.

[46] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[47] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[48] L. Molnár, G. Pongrácz, G. Enyedi, Z. L. Kis, L. Csikor, F. Juhász, A. Kőrösi, and G. Rétvári, "Dataplane specialization for high-performance OpenFlow software switching," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 539–552.

[49] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *Proc. ACM SIGCOMM Conf.*, Aug. 2011, pp. 254–265.

[50] V. Jeyakumar, M. Alizadeh, Y. Geng, C. Kim, and D. Mazières, "Millions of little minions: Using packets for low latency network programming and visibility," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 3–14, 2014.

[51] B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge, "Smart packets: Applying active networks to network management," *ACM Trans. Comput. Syst.*, vol. 18, no. 1, pp. 67–88, Feb. 2000.

[52] P. Benáček, V. Puš, H. Kubátová, and T. Čejka, "P4-To-VHDL: Automatic generation of high-speed input and output network blocks," *Microprocessors Microsyst.*, vol. 56, pp. 22–33, Feb. 2018.

[53] *P4 Language Consortium*. Accessed: Dec. 20, 2022. [Online]. Available: https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html

[54] D. Ding, M. Savi, F. Pederzolli, and D. Siracusa, "Design and development of network monitoring strategies in P4-enabled programmable switches," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Apr. 2022, pp. 1–6.

[55] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, vol. 10, no. 8, San Jose, CA, USA, 2010, pp. 89–92.

[56] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "HULA: Scalable load balancing using programmable data planes," in *Proc. Symp. SDN Res.*, Mar. 2016, pp. 1–12.

[57] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 503–514.

[58] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 15–28.

[59] H. Harkous, M. Jarschel, M. He, R. Pries, and W. Kellerer, "P8: P4 with predictable packet processing performance," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 3, pp. 2846–2859, Sep. 2021.

[60] E. O. Zaballa, D. Franco, M. Zang, A. Atutxa, J. Sasiain, A. Pruski, E. Rojas, M. Higuero, and E. Jacob, "Data plane latency analysis and prediction modeling in P4 programmable asic-based hardware," Jan. 2023. [Online]. Available: https://ssrn.com/abstract=4348218, doi: 10.2139/ssrn.4348218.

[61] Y. Chen, L. Yen, W. Wang, C. Chuang, Y. Liu, and C. Tseng, "P4-enabled bandwidth management," in *Proc. 20th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Sep. 2019, pp. 1–5.

[62] O. Fernando, H. Xiao, and X. Che, "Evaluation of underlying switching mechanism for future networks with P4 and SDN (workshop paper)," in *Proc. Int. Conf. Collaborative Comput., Netw., Appl. Worksharing*. London, U.K.: Springer, 2019, pp. 549–568.

[63] J. Alvarez-Horcajo, I. Martínez-Yelmo, D. Lopez-Pajares, J. A. Carral, and M. Savi, "A hybrid SDN switch based on standard P4 code," *IEEE Commun. Lett.*, vol. 25, no. 5, pp. 1482–1485, May 2021.

[64] H. T. Dang, M. Canini, F. Pedone, and R. Soulé, "Paxos made switchy," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 2, pp. 18–24, Apr. 2016.

[65] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, and J. Rexford, "PISCES: A programmable, protocol-independent software switch," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 525–538.

[66] A. AlSabeh, J. Khoury, E. Kfoury, J. Crichigno, and E. Bou-Harb, "A survey on security applications of P4 programmable switches and a STRIDE-based vulnerability assessment," *Comput. Netw.*, vol. 207, Apr. 2022, Art. no. 108800.

[67] E. O. Zaballa, D. Franco, M. S. Berger, and M. Higuero, "A perspective on P4-based data and control plane modularity for network automation," in *Proc. 3rd P4 Workshop Eur.*, Dec. 2020, pp. 59–61.

[68] F. Hauser, M. Schmidt, M. Häberle, and M. Menth, "P4-MACsec: Dynamic topology monitoring and data layer protection with MACsec in P4-based SDN," *IEEE Access*, vol. 8, pp. 58845–58858, 2020.

[69] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A survey on data plane programming with P4: Fundamentals, advances, and applied research," *J. Netw. Comput. Appl.*, vol. 212, Mar. 2023, Art. no. 103561.

[70] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim, "Language-directed hardware design for network performance monitoring," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 85–98.

[71] G. Gibb, G. Varghese, M. Horowitz, and N. McKeown, "Design principles for packet parsers," in *Proc. Architectures Netw. Commun. Syst.*, 2013, pp. 13–24.

[72] R. Mittal, R. Agarwal, S. Ratnasamy, and S. Shenker, "Universal packet scheduling," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2016, pp. 501–521.

[73] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "NetCache: Balancing key-value stores with fast in-network caching," in *Proc. 26th Symp. Operating Syst. Princ.*, Oct. 2017, pp. 121–136.

[74] S. Buzura, M. Lehene, B. Iancu, and V. Dadarlat, "An extendable software architecture for mitigating ARP spoofing-based attacks in SDN data plane layer," *Electronics*, vol. 11, no. 13, p. 1965, Jun. 2022.
[75] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, and R. Clark, "Kinetic: Verifiable dynamic network control," in *Proc. 12th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2015, pp. 59–72.

BHARGAVI GOSWAMI received the Ph.D. degree in computer science from NIMS University, India, in 2015. She was an Assistant Professor and an Associate Professor with Indian universities. She is currently with the Queensland University of Technology, researching reducing the latency of communication networks for smart grids besides performing academic activities. She is a Profound Researcher and an experienced Academician. Her research interests include software-defined networks, smart grid communications, and large-scale corporate communication networks.

MANASA KULKARNI received the M.Sc. degree in computer science from CHRIST (Deemed to be University), Bengaluru, India, in 2020, where she is currently pursuing the Ph.D. degree in computer science. Her research interests include software-defined networks, P4 language, Pyretic and Frenetic, and SDN controllers.

JOY PAULOSE received the M.Sc. degree in physics with electronics from Mahatma Gandhi University, Kerala, in 1986, the M.Tech. degree in electronics engineering from the Cochin University of Science and Technology, and the Ph.D. degree in networks from CMJ University. He was in the education industry for more than three decades. He has been with CHRIST (Deemed to be University), Bengaluru, where he is currently the Head of the Department of Computer Science and Statistics. Since 2013, he has been involved with research as part of the active group. His research interests include smart networks, software-defined networks, image processing, and electronics.