IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 22, 2019, accepted April 2, 2019, date of publication April 11, 2019, date of current version April 19, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2910140

A Survey on Data Plane Flexibility and Programmability in Software-Defined Networking

ENIO KALJIC, (Member, IEEE), ALMIR MARIC, (Member, IEEE), PAMELA NJEMCEVIC, AND MESUD HADZIALIC, (Member, IEEE)
Department of Telecommunications, Faculty of Electrical Engineering, University of Sarajevo, 71000 Sarajevo, Bosnia and Herzegovina

Corresponding author: Enio Kaljic (enio.kaljic@etf.unsa.ba)

This work was supported by the Government of Federation of Bosnia and Herzegovina, Federal Ministry of Education and Science, under Grant 05-39-2565-1/18.

ABSTRACT Software-defined networking (SDN) has attracted the attention of the research community in recent years, as evidenced by a large number of survey and review papers. The architecture of SDN clearly recognizes three planes: application, control, and data planes. The application plane executes network applications, the control plane regulates the rules for the entire network based on the requests generated by network applications, and based on the set rules, the controller configures the switches in the data plane. The role of the switch in the data plane is to simply forward packets based on the instructions given by the controller. By analyzing the SDN-related research papers, it is observed that research, from the very beginning, is insufficiently focused on the data plane. Therefore, this paper gives a comprehensive overview of the data plane survey with a particular emphasis on the problem of programmability and flexibility. The first part of the survey is dedicated to the evaluation of actual data plane architectures through several definitions and aspects of data plane flexibility and programmability. Then, an overview of the SDN-related research was presented with the aim of identifying the key factors influencing the gradual deviation from the original data plane architectures given with ForCES and OpenFlow specifications. In this paper, we used the term data plane evolution for this deviation. By establishing a correlation between the treated problem and the problem-solving approaches, the limitations of ForCES and OpenFlow data plane architectures were identified. Based on the identified limitations, a generalization of approaches to addressing the problem of data plane flexibility and programmability has been made. By examining the generalized approaches, open issues have been identified, establishing the grounds for future research directions proposal.

INDEX TERMS Data plane, data plane abstractions, data plane architectures, data plane flexibility, data plane implementations, data plane languages, data plane programmability, deeply programmable networks, description languages, energy consumption, energy efficiency, hardware-based implementations, measurement, monitoring, network virtualization, network functions virtualization, networking technologies, performance, programming languages, quality of service, reliability, security, software-based implementations, software-defined networking, stateful data plane.

I. INTRODUCTION

Traditional communication networks are constructed from a large number of network devices that perform various tasks such as switching, routing, maintaining quality of service, monitoring and management, ensuring security and reliability, etc. To respond to these challenges, complex network algorithms and protocols are implemented on these network devices, which in the majority of cases are proprietary and implemented in the form of closed code. Maintenance and management of such networks is achieved by particular configuration of network devices through interfaces that vary from one vendor to another. Standardization of network interfaces and protocols, aimed at unifying the

The associate editor coordinating the review of this manuscript and approving it for publication was Nitin Nitin.

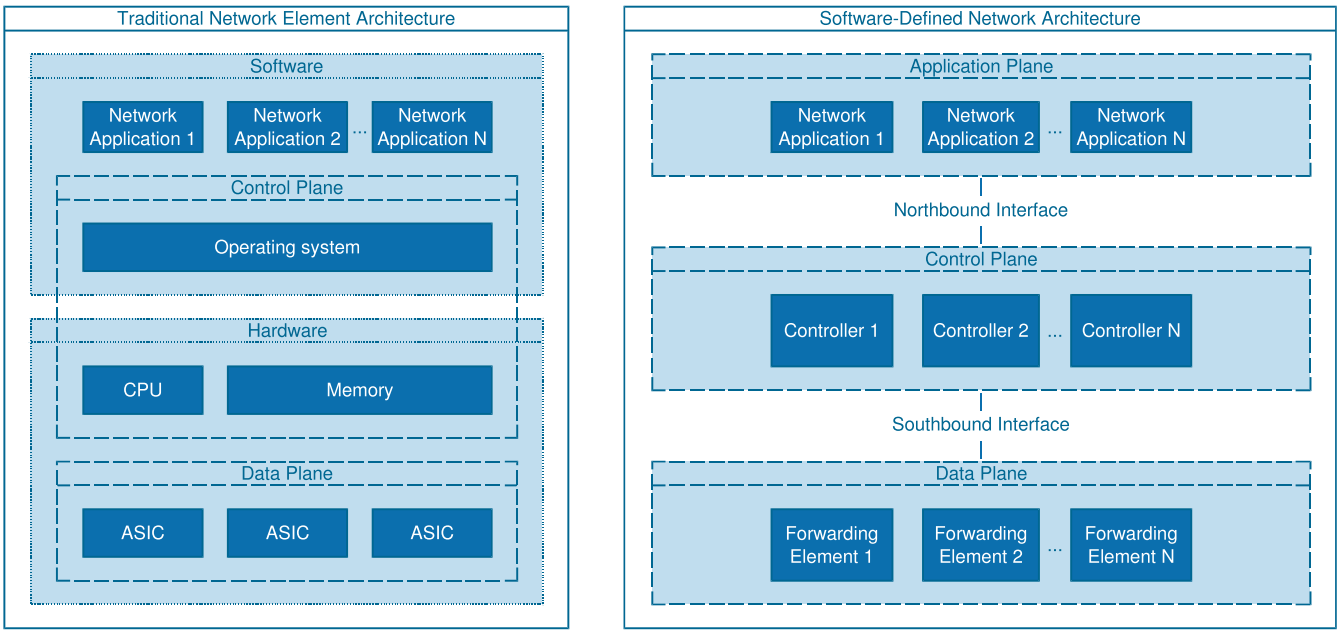


FIGURE 1. Migration from traditional to software-defined network architecture.

network ecosystem and increasing the degree of interoperability, is a complex and time-consuming process. This approach has slowed down innovation and increased complexity and cost of maintenance and network management. In response to this problem, researchers in the 1990's [1], [2] have applied the analogy of relatively simple reprogramming of classical computers to computer networks, thus establishing the basis for the development of active networks [3]. The active network concept is based on the fact that the packet carries the program instead of raw data (ie., smart packets [4]). Network devices, when they receive a smart packet, execute the program it carries, and in accordance with the data plane design carry out different actions on the packet. In this way, network devices create an environment that responds to what the packet carries instead of passively transmitting packet payload from one node to another.

In the early 2000s, the idea of network programmability, which comes from active networks, is articulated by separating the control and data plane, thus creating the concept of software-defined networking (SDN) [5]. Figure 1 illustrates the transition from a traditional network architecture to a software-defined network. In the backbone of conventional network architecture, there is a networking device which performs all control and data plane tasks using a hard separable combination of software and hardware. On the other hand, in the SDN, the entire network intelligence is centralized in the application and control plane, where the application plane is composed of different network applications, and one or more controllers make the control plane. Network applications are performing routing algorithms, quality of service (QoS) mechanisms, control and network management mechanisms, etc., and are generating rules, according to which the network traffic should be treated. Generated rules are delivered to the control plane via a specially defined northbound interface. Based on these rules, controllers make specific forwarding rules and, according to them, configure packet switches via a southbound interface. In the end, network devices (routers and switches), in the data plane, perform a simple forwarding of packets based on a quick lookup of the forwarding tables.

The standardization of the first SDN architecture was started through the Forwarding and Control Element Separation (ForCES) requirements specification by the IETF in 2003 in RFC3654 [6] and a year later confirmed in RFC3746 [7]. According to the ForCES specification given in RFC3746, the network element (NE) consists of several control elements (CE) in the control plane and the forwarding elements (FE) in the data plane. Since ForCES was not designed with a long-term vision to implement the SDN architecture, only with the emergence of OpenFlow [8], the significance and usefulness of SDN architecture has arisen. OpenFlow is based on an Ethernet switch with an internal flow table and a standardized interface for adding and deleting records in the flow table. With understanding the need for standardization of communication interfaces and protocols, IETF expands the ForCES specification with RFC5810 [9]. Although both ForCES and OpenFlow follow the same idea of the control and data plane separation, they are technically different from the aspects of architecture and forwarding model, as analyzed in [10].

The development of SDN has attracted the attention of the research community in recent years, as evidenced by the large

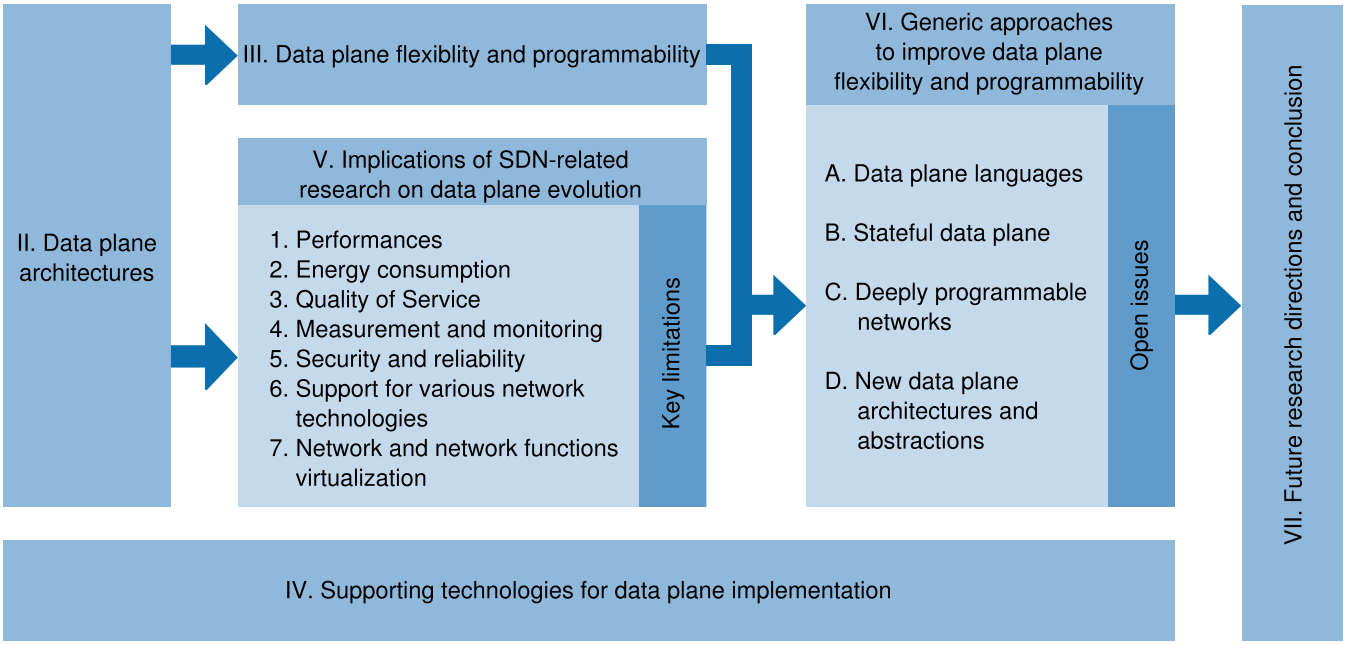


FIGURE 2. Survey methodology.

number of review and survey papers. General overviews of SDN research are provided in [11]–[15], while in a significant number of papers, targeted reviews were made by the issue addressed within the SDN. Thus, the development of traffic engineering in OpenFlow based networks is shown in [16], while a review of new challenges in the development of SDN-based traffic engineering is given in [17]. Special attention was given to the review of QoS models and mechanisms in [18]. In [19], problems and solutions related to scalability, reliability, security, and performance of SDN were analyzed, while in [20], [21], issues of energy efficiency and energy consumption in SDN were discussed. An overview of the research experimentation platforms is given in [22].

By analyzing the survey papers mentioned above, it has been noted that the SDN research has been focusing on the control and application plane programmability from the very beginning. For the data plane, ever since the inception of the SDN idea, it was considered that it should follow two basic principles:

1) simplicity that is seen in the process of packets forwarding in data plane, and
2) the generality that is indicated in the independence of the SDN architecture from the technology through which the network is implemented.

Since the main challenges of SDN have been in the control plane, that led to the neglect of data plane development. Therefore, the focus of this paper is on the data plane, especially in term of its flexibility and programmability.

Considering the different definitions of the data plane flexibility [23]–[25] and programmability [4], [26]–[30], we advocate that flexibility denotes the possibility of a data plane to timely respond to new conditions in the network, and programmability a method by which flexibility is achieved. Under new conditions in the network, we include changes in requirements, constraints, and data plane state.

Although, specific issues of data plane flexibility and programmability were addressed in above-mentioned papers, it is important to note that there is no comprehensive survey of data plane research from the aspect of its flexibility and programmability. It is also important to emphasize that although there are review papers that dealt with software-defined wireless networks [31]–[33], data plane of the wireless network is out of the scope of this paper because it is being implemented using the Software-Defined Radio (SDR) techniques.

Consequently, the aim of this paper is a survey of data plane research in a wired SDN, which appropriately addresses the problems of programmability and flexibility, and establishes preconditions for the advancement of its development through a proposal of future research directions.

To accomplish this goal, several tasks have been carried out according to the methodology presented in Figure 2, which are at the same time the outline of this paper. At first, an overview of the data plane architecture in ForCES and the OpenFlow-based SDN, reflecting on the historical context of development and the differences between these two models is given in Section II. Afterwards, in Section III is given an overview of the definitions of network flexibility and programmability and some general considerations of flexibility in other domains. Then, a review of the constraints of ForCES and OpenFlow-based data plane architectures, through the considered definitions and aspects of flexibility and programmability, is presented. Given that a lot of the data

plane research, discussed in Sections II, V, and VI, is established on the experimental evaluation, in Section IV is given an overview of hardware- and software-based technologies which served as good support for data plane implementation. In Section V is given an overview of SDN-related research whose results have implied a data plane evolution. Under the data plane evolution, we indicate a gradual deviation from the original data plane architectures given with ForCES and OpenFlow specifications, resulting in the need to address the problem of programmability and the flexibility of the data plane in a more generic way. By reviewing the research which had addressed different problems in seven categories, as shown in Figure 2, we observed several common problem-solving approaches. Then, by establishing the correlation between treated problems and problem-solving approaches, we identified the key limitations of ForCES and OpenFlow data plane architectures. Based on identified key limitations in Section V and discussed aspects of flexiblity and programmability in Section III, we generalized approaches to improving the data plane flexibility through four methods for improvement of the data plane programmability. Based on critical review of generic approaches to improve data plane flexibility and related open issues, future research directions are proposed in Section VII.

The main contributions of this paper can be summarized as following:

- An overview of supporting hardware and software technologies, which enabled SDN's data plane implementation, is given from the perspective of several definitions of flexibility and programmability.
- Key limitations in term of flexibility and programmability of OpenFlow and ForCES based data plane architectures are identified by comprehensive review of SDN-related research.
- The proposed future research directions are established on the basis of identified open issues of generic approaches to data plane flexibility improvement.

Table 1 shows the list of abbreviations in alphabetical order which are used more than once throughout the paper or outside the same paragraph.

TABLE 1. List of abbreviations.

Abbreviation	Full phrase
ARP	Address Resolution Protocol
ATCA	Advanced Telecommunications Computing Architecture
BSV	Bluespec System Verilog
CE	Control Element
CPU	Central Processing Unit
DCN	Data Center Network
DDoS	Distributed Denial-of-Service
DDR	Double Data Rate
DOCSIS	Data Over Cable Service Interface Specification
DPDK	Data Plane Development Kit
DPI	Deep Packet Inspection
DPN	Deeply Programmable Network
DRAM	Dynamic Random-Access Memory
FE	Forwarding Element
FIFO	First-In-First-Out
ForCES	Forwarding and Control Element Separation
ForTER	ForCES-based Router
FP	Forwarding Processor
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GPON	Gigabit Passive Optical Network
HAL	Hardware Abstraction Layer
IoT	Internet of Things
IP	Internet Protocol
I/O	Input/Output
LFB	Logical Functional Block
MAC	Medium Access Control
MCF	Multi-Core Fiber
MPLS	MultiProtocol Label Switching
NAT	Network Address Translation
NE	Network Element
NFV	Network Function Virtualization
NIC	Network Interface Card
OCS	Optical Circuit Switching
OPP	Open Packet Processor
OPS	Optical Packet Switching
OVS	Open vSwitch
PCI(e)	Peripheral Component Interconnect (Express)
PE	Processing Element
PHY	Physical Layer
POF	Protocol-oblivious Forwarding
ROADM	Reconfigurable Optical Add-Drop Multiplexer
QoS	Quality of Service
RISC	Reduced Instruction Set Computer
RLDRAM	Reduced Latency DRAM
RMT	Reconfigurable Matching Table
SFP	Small Form-factor Pluggable
SMF	Single-Mode Fiber
SoC	System on a Chip
SoFPGA	System on FPGA
SDN	Software-Defined Network(ing)
SRAM	Static Random-Access Memory
TCAM	Ternary Content-Addressable Memory
TCP	Transport Control Protocol
ToR	Top of the Rack
UDP	User Datagram Protocol
VM	Virtual Machine
VNF	Virtual Network Function
XFSM	eXtended FSM

II. DATA PLANE ARCHITECTURES

An overview of the data plane architecture in ForCES and the OpenFlow based SDN is given in this section. In addition, some architectures inspired by ForCES have been reviewed, and at the end of the section, a review of the differences between the ForCES and OpenFlow data plane architectures is provided.

The first proposal of the data plane architecture was specified by RFC5812 [34], according to which the resources within ForCES FE are represented by logical functional blocks (LFBs), as illustrated in Figure 3. LFB is a logical representation of a single packet processing functionality. The data paths through which the packets pass are formed by the interconnection of multiple LFBs, and they enable complex tasks execution during packet processing. Definitions and implementations of 22 LFBs in accordance with ForCES specification are presented in [35]. In addition to the definition of LFBs, the definition of eight frame types, 43 data types, and 14 metadata types are given. Metadata are

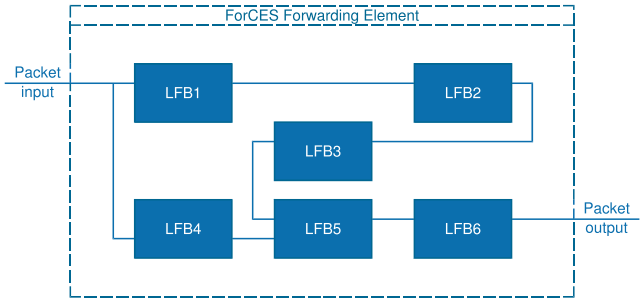


FIGURE 3. ForCES forwarding element architecture.

associated with packets when traveling between LFBs within the network element. The proposed definitions covered basic Internet protocol version 4 (IPv4) packet processing functionality, while the definitions of advanced elements required to support QoS functionality or firewall functionality were beyond the scope of the above-mentioned paper. The design and implementation of the ForCES protocol are presented in [36]. The introduced implementation enables the creation of new LFB test topologies within the network element structure, which makes it a useful tool in the research of the ForCES data plane.

The LFB chain composition method, which allows the LFB series to combine in the LFB chain according to the application, is presented in [37]. LFB chaining method is divided into three types: (1) sequential chaining, (2) chaining of branches and (3) a hybrid chaining. In the mentioned work, a method for sequential chaining was proposed, and a work frame for that method was implemented. The framework consists of an LFB chain matching agent, which generates an appropriate LFB chain based on the incoming request. Agent performs the matching process in three steps: (1) mapping requests to a set of LFBs that can respond to a given request, (2) combining LFBs into one or more chains, and (3) selection of the best chain from a set of chains. An overview of research in the field of development and application of the ForCES specification is given in [38]. It has been noted that ForCES provides a significant support in various areas where distributed packet processing on network elements is required. The possibility of realization of custom-defined LFBs makes the ForCES model very flexible and powerful. Also, the vision of using ForCES in the unification of all network technologies, such as optical networks, wireless networks and Internet of Things (IoT), is presented. Although ForCES was presented as a promising model, it was concluded that its application is not at a satisfactory level due to several factors of which the most significant are: (1) lack of model adoption in the network industry, and (2) little use in academic environment due to lack of support for experimental work in the form of usable open source code.

Inspired by ForCES, two architectures based on the idea of separating the control and data plane, NEon and Ethane, are presented in [39], [40]. NEon [39], in accordance with the SDN principles, consists of two planes: (1) control plane policy manager, and (2) programmable rule enforcement

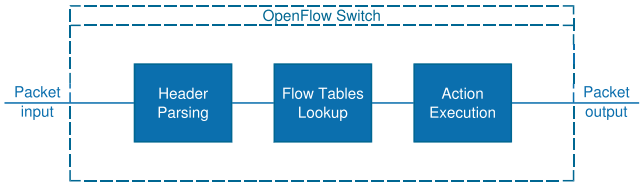


FIGURE 4. OpenFlow switch architecture.

device (PRED). PRED has been realized as a high-performance programmable machine for packets classification and actions execution. It consists of logical functional blocks aggregation that enables the implementation of different network services. Logical functional blocks are providing packet processing functionalities such as flow identification, packet classification, and action processing. Action processing was achieved by using programmable dispatch machines that allow packet data manipulation. Ethane [40] is an enterprise network architecture, consisting of: (1) a centralized controller that defines network policies for all packets, and (2) a group of simple Ethane switches. The Ethane switch contains a secure channel to the controller and a simple flow table. Packets arriving at the Ethane switch are forwarded based on the records in the flow table. In the absence of an appropriate record, the packet is forwarded to the controller together with information about where the packet came from. In this case, the controller has the task of defining the forwarding rule for that packet and updating the flow table on the switch. The records from the Ethane flow table contain: (1) header according to which the matching is performed with the headers of incoming packets, (2) action that tells the switch to what to do with the packet, and (3) additional data related to the flow (various counters). Header fields cover transport control protocol (TCP), user datagram protocol (UDP), IP and Ethernet protocol, and physical port information. Supported actions are: (1) forwarding the packets to the corresponding interface, (2) update of byte and packet counters, and (3) setting the flows activity bits. Beside listed, additional actions are possible such as placing packets in different queues or changing packet headers. In Ethane network architecture, all switches do not have to be Ethane switches, enabling a gradual migration from classic networks to Ethane-based networks.

On the other hand, the OpenFlow data plane [8] consists of fixed architecture switches made up of: (1) flow table containing flow records with associated actions, (2) a secure channel to the OpenFlow controller, and (3) OpenFlow protocol that provides standardized communication between switches and controllers. OpenFlow switch working principle, illustrated in Figure 4, is based on simple forwarding of the packet between the ports based on the records in the flow table defined by the remote control process, i.e. the controller. Each record in the flow table contains three fields: (1) the packet header which defines the flow, (2) an action that specifies how packets are processed, and (3) statistics related to the flow (e.g., packet and byte count). OpenFlow switch has to support

at least the following three actions: (1) forwarding a packet to a specific port or multiple ports, (2) packet encapsulation and forwarding to the controller, and (3) packet discarding. The encapsulation of the packet and sending to the controller is performed only for the first packet of the new flow, i.e., when for the incoming packet there are no records present in the flow table. Then, the controller generates the forwarding rule for the new flow and delivers it to the switch, allowing the switch to autonomously handle all subsequent packets from that flow.

An overview and the analysis of the differences between ForCES and OpenFlow regarding architecture and forwarding model are given in [10]. ForCES architecture implies separating the control and data plane within a single network element. This is achieved on two levels: (1) the first level implies communication separation in the sense that CE and FE are using a standard protocol instead of proprietary interfaces, and (2) physical separation that allows CEs and FEs to be executed in physically separated devices, and together form a network element. Interfaces between two ForCES NEs are the same as between standard switches and routers, so ForCES NEs can be transparently connected to existing conventional network elements. The ForCES control functions are still based on the use of distributed protocols. However, OpenFlow separates the control and data planes in such a way that the data plane consists of more simple switches and the control plane of the whole network is made up from one centralized controller. The OpenFlow architecture supports two types of switches:

1) ''pure'' OpenFlow switch - contains only a data plane based on flow tables, and provides an interface to a logically centralized controller. Logically centralized controller performs all control tasks such as: (1) collecting data on network operation and making decisions according to management logic, (2) installing rules on switches' flow tables through the OpenFlow protocol, and (3) providing an open application programming interface (API) to user applications.
2) hybrid switch - also supports autonomous operation as a conventional Ethernet switch (eg., in the absence of OpenFlow controller).

The forwarding model in the ForCES data plane is based on packet processing through the LFB composition described by the directed graph. Each LFB defines a straightforward operation which is performed on the packet that passes through it. Typical examples of LFBs are: IPv4 Longest Prefix Matching, address resolution protocol (ARP), Internet control message protocol (ICMP), Queue, Scheduler, etc. However, with OpenFlow, the forwarding model is based on the flow tables manipulations. OpenFlow switches handle packets with flow granularity. Therefore, some standard network functionality that is run on the packet level (e.g., ARP) is very difficult to implement with OpenFlow. On the other hand, using the ForCES architecture, it is possible to define LFBs whose work principle is similar to OpenFlow flow tables, which confirms the flexibility of ForCES forwarding model. Thus, the possibility of extending OpenFlow architecture with ForCES concepts is explored in [41]. By comparing the data plane of the OpenFlow and ForCES architectures, it has been observed that ForCES elements can describe some aspects of OpenFlow: (1) suitable LFB components can describe packet header fields lookup, counters, and actions, (2) unique attributes of LFBs can describe the set of supported actions and the mode of their execution, and (3) directed graph of LFBs can describe OpenFlow pipeline. In regard to the observation, the middleware based on the ForCES wrapper around the OpenFlow switch, which would allow switch control via the ForCES control element or OpenFlow controller, was proposed.

III. DATA PLANE FLEXIBILITY AND PROGRAMMABILITY

An overview of network flexibility definitions and general considerations of flexibility in other domains, which can be useful in valuing data plane flexibility, is given in this section. Then, an overview of the definition of programmability and the connection between the programmability and the flexibility of the data plane is presented. Finally, a review of the limitations of ForCES and OpenFlow based data plane architectures, from the perspective of the described definitions of flexibility and programmability, is given.

A. DEFINITION OF FLEXIBILITY

Since there is no common approach to the definition of network or data plane flexibility, various definitions have been used and proposed in many papers. While some observed flexibility through the structure and design of the system [23]–[25], [42]–[44], others tied the definition of flexibility to the resilience of the system [45]–[47].

1) FLEXIBILITY IN SDN

Although the benefit of SDN paradigm is in the development flexibility of new control logic, in [28] is emphasized the importance of reconfiguration flexibility which allows the addition of support for new protocols and flexibility of the data plane structure. According to papers [23] and [24] which have dealt with the flexibility of softwarized networks, which include SDN, flexibility is defined as the ability of the network to adapt its resources, such as flows and topology, to changes in the requirements placed to the network. Adaptation implies changing the network topology, configuration and position of network functions. Given that there is no generally accepted definition of flexibility, the following aspects of flexibility are proposed in [23]:

- flow steering,
- function placement,
- function scaling,
- function operation.

From the aspect of flow steering, an element, which supports both packet forwarding and copying, is more flexible regarding an element which supports only packet forwarding. Flexibility from the aspect of function placement reflects in the ability to dynamically change the position of

functions during network operation, while the granularity of the resource allocation between the functions affects the aspect of scalability. The configurability and programmability of the network element functionalities are covered through the aspect of the function operation. Additionally, an aspect of topology adaptation, in term of adding or deleting links and nodes, is introduced in [24].

Network flexibility was also a subject of research presented in [25], where it is defined as ''timely support of changes in the network requirements.'' Under changes in network requirements, they include traffic variation, user mobility, network lease, network upgrades, and failure mitigation. In the aforementioned papers, they have considered two cases in which the network can support these requirements:

1) a network design which allows meeting the requirements without adaptation, and
2) an adaptation of topology, flows, resources, and functions.

In the second case, adaptation should be carried out within the given time constraints.

2) FLEXILIBITY IN OTHER DOMAINS

Flexibility can be observed through the inclusion of different options or design alternatives in the system structure, which was the basic idea of [42]. According to the proposed approach, the system can be designed using nodes and links, where each node represents the decision in the design process, and the link indicates logical or temporal dependence. According to cited paper, the system flexibility manifests in the total number of paths from the source node to the destination node, where each path represents a sequence of decisions leading to the fulfillment of the set requirement. Similarly, in [43] flexibility is defined via the ratio of the number of different paths and the total number of nodes.

The impact of layer abstraction and modular decomposition on the flexibility of the system is explored in [44]. The analysis was carried out for four design strategies:

1) integral system design in which one form shares functions, including unused functions,
2) modular system design in which multiple forms are mapped to multiple forms by a one-to-one principle,
3) layered system design in which all functions are included even not used (latent functions), and
4) synergistic system design which includes latent functions but also allows adding new ones as needed.

Using the simulation techniques in [44] has been shown that both system design strategies contribute to the flexibility of the system, and their contribution is additive, making the synergistic system design the most flexible.

In addition to the flexibility definitions which are related to the structure and design of the system, in [45]–[47] flexibility is observed through resilience, defined as the ability of the system to recover quickly from external or internal disruptions and return to the state of equilibrium. In this context, the disruption can be modeled as a new requirement set to the system, and the ability of the system to respond to new requirements and continue with the correct functioning as a system resilience.

B. DEFINITION OF PROGRAMMABILITY

Data plane programmability has been in the focus of research presented in [4], [26]–[30]. Programmability has been observed in [4] as a significant characteristic of the network through the level of programmability indicating the method, the granularity and the time scale in which new functionalities can be introduced into the network infrastructure.

In [26] and [27] researchers advocate that data plane programmability reflects in its depth and the way of its implementation. The depth of programmability they see through management capabilities of processes below the level of packet forwarding, which includes caching, transcoding, support for new protocols, and so on. Regarding the method of data plane implementation, a data plane which is implemented entirely in the software is programmable, and one implemented in hardware is non-programmable. Contrary to this, in [28] researchers believe that the data plane programmability can be achieved with the use of reconfigurable hardware and convenient programming languages. Regardless of the implementation method, in [29] it is deemed that data plane programmability can be seen in stateful flow processing.

A comprehensive definition of data plane programmability is given in [30], according to which programmability implies the switch capability to expose the packet processing logic to the control plane to support systematic, fast and comprehensive reconfiguration.

From the considered definitions of flexibility and programmability, we see data plane programmability as a key factor in achieving flexibility from the aspects of adaptation of topology, flows, functions and resources.

C. FLEXIBILITY AND PROGRAMMABILITY OF FORCES AND OPENFLOW

According to the specification given in RFC5812 [34], the data plane of ForCES consists of FEs presented by interconnected LFBs. Therefore, with the adequate support for the programmability of individual LFBs and their arrangement into arbitrary functional topologies, the data plane of ForCES can be viewed as highly flexible from the aspect of adaptation of functions.

On the other hand, the data plane of OpenFlow is fixed pipeline structure whose forwarding model is based on the lookup of flow tables and execution of associated actions. Programmability of the data plane of OpenFlow is limited to the level of table flow content, which restricts its flexibility solely to the aspect of flows adaptation.

Since neither ForCES nor OpenFlow based data plane architectures are flexible enough in term of considered aspects, a significant number of research has gone in the direction of data plane evolution to adequately respond to various functional requirements. Section V is dedicated to the review of that research with the aim of identifying the

key limitations of ForCES and OpenFlow based data plane architecture in term of flexibility and programmability.

IV. SUPPORTING TECHNOLOGIES FOR DATA PLANE IMPLEMENTATIONS

Given that a lot of research, discussed in Sections V and VI, is based on experimental verification of proposed data plane architectures, this section presents an overview of hardware- and software-based technologies which served as a good platform for their implementations. At first, hardware architectures were used to implement packet switching nodes in the SDN, but slightly later software architectures were also used due to processing power limitations of conventional computer systems no longer being an issue. The processing power of today's computer systems has reached a significant level which enables the implementation of packet switching nodes whose performance is comparable to hardware-based implementations, placing the software-based implementations in an equally prominent position.

A. HARDWARE-BASED IMPLEMENTATIONS

By reviewing the research that dealt with hardware-based implementation of packet switching nodes and its application in SDN's data plane, we perceived the following categories:

1) field-programmable gate array (FPGA) based implementations,
2) system on a chip (SoC) based implementations,
3) network processor (NP) based implementations,

1) FPGA-BASED IMPLEMENTATIONS

NetFPGA - the first FPGA-based platform, specially designed to teach network equipment development, is presented in [48], [49]. The first version of the NetFPGA board contains three Altera EP20K400 APEX devices, three 1MByte static random-access memory (SRAM) chips and 8-port Ethernet controller. One of three FPGA chips, called Control FPGA (CFPGA), is pre-programmed and connects two user FPGA chips (UFPGA) to an Ethernet controller. All communication on the board takes place via Ethernet ports. Although the board does not have a central processing unit (CPU), its operation is possible thanks to the virtual network system, the software executed on the computer where the card is embedded. The software can access the hardware registers, using a dedicated Ethernet frame with the CFPGA being responsible for its decoding and execution.

The development of new versions of the NetFPGA board proceeded because of the limitations of the first version such as: impractical size of the board which can be only fitted into specially designed computer chassis, low speed - the first version had eight 10Mbps Ethernet ports, and lack of processor. Thus, in [50], [51], NetFPGA-v2 and NetFPGA-v2.1 boards are presented. NetFPGA-v2 is made as a 32-bit full-length peripheral component interconnect (PCI) board running at 33MHz. The board is equipped with a Xilinx Spartan chip through which PCI communication is supported and the Xilinx V2P30 chip to which the user design is programmed. The UFPGA has two 512Kx36 SRAMs and is connected to the Marvell Quad 10/100/1000 Ethernet physical layer (PHY) chip through the standard gigabit media-independent interface (GMII). NetFPGA-v2.1 brings two additional DDR3 synchronous dynamic random-access memory (SDRAM) chips that work asynchronously with the UFPGA chip. The standard NetFPGA library contains a skeleton of Verilog design that instantiates four Gigabit Ethernet Media Access Controllers (GMAC) and interfaces to SRAM and DDR2 memory. User design is implemented as a pipeline following the standard request-grant first-in-first-out (FIFO) protocol. The pipeline consists of input modules, user filter, and the output module. Input modules are connected to four Gigabit Ethernet network interfaces and host processors via a PCI interface. The user filter performs tasks such as decapsulation, decryption, and other user-defined functions. The output module performs an output port lookup to determine the port to which the packet must be forwarded. For example, the Ethernet switch or IP router logic are mainly implemented in the output module of the pipeline.

The first application of FPGA technology in the SDN's data plane is the Ethane switch implementation [52]. The data plane of the switch is implemented on the NetFPGA-1G board as a pipeline with two exact match flow tables, one for the packets to be forwarded and the other for the packets to be discarded. Packets that do not match the records in flow tables are forwarded to the software responsible for maintaining the flow table through record addition and deletion.

Considering the growing need for fast prototyping platforms in the forthcoming period, a 40Gbps PCI Express card with a Xilinx Virtex-5 chip, called NetFPGA-10G, is presented in [53]. NetFPGA-10G has four 10Gbps Ethernet interfaces in SFP+ form that are connected via additional PHY transceivers to the FPGA, and RLDRAMII and QDRII memory controllers for additional SRAM and DRAM memory. The Open Component Portability Infrastructure (OpenCPI) interface is used to connect the NetFPGA-10G card via PCIe to a computer. The AMBA4 AXI-Stream interface is used for packet transmission within the reference design and the AMBA4 AXI-Lite interface for signaling.

NetFPGA-1G-CML and NetFPGA-SUME, featured in [54], [55], are the successors of NetFPGA-1G and NetFPGA-10G platforms based on the 7th generation of Xilinx FPGAs. NetFPGA-1G-CML board is based on the Xilinx Kintex-7 FPGA. Improvements compared to NetFPGA-1G are reflected in:

- three times more of FPGA logical elements,
- four times more of Block RAM capacity,
- 512 MB DDR3 instead of 64 MB DDR2,
- additional 4.5 MB QDRII+ memory,
- 4x Gen. 2 PCIe instead of PCI.

It is compatible with Stanford's 10G architecture design, which allows relatively easy portability of designs from NetFPGA-10G platforms. NetFPGA-SUME is based on the Xilinx Virtex-7 960T FPGA chip containing 30 serial 13.1Gbps transceivers through which the FPGA chip is

connected to four 10Gbps SFP+ Ethernet interfaces, PCIe, and two expansion connectors through which multiple NetFPGA-SUME boards can be interconnected. Therefore, no additional physical layer controller is required for the implementation of 10G Ethernet applications, which is the most significant improvement compared to the NetFPGA-10G board. Through the expansion connector, it is possible to implement a 100Gbps switch, and by the interconnection of multiple cards, it is possible to produce a 300Gbps non-blocking switch, which makes this platform suitable for the research of high-throughput networks.

The multi-purpose highly-programmable network platform based on FPGA technology, called C-GEP, is presented in [56]. C-GEP enables flexible implementation of network nodes that support different application types such as SDN switches, media gateways, traffic generators, deep packet inspection (DPI), etc. All of these applications are possible over 1, 10, 40 and 100 Gbps traffic. The Virtex-6 FPGA chip on the C-GEP motherboard performs packet forwarding tasks, while the embedded COM Express PC is responsible for management and SDN control functionality implementation. Installation of the appropriate firmware defines the architecture of the network device implemented on C-GEP.

OpenFlow switch implementation on NetFPGA - which can store more than 32,000 flow records and performs at the speed of four NetFPGA 1G ports, is described in [57]. The switch is made of software and hardware components. The software component of the switch is from the user space responsible for communication to the OpenFlow controller, and from the kernel space to the table flow maintenance, packet processing, and statistics update. The hardware component of the OpenFlow switches implements a different output port lookup as compared to the reference router and uses dynamic random-access memory (DRAM) for output queues.

A good foundation for the future implementations of SDN's data plane using the NetFPGA platform was provided by the reference architecture of the packet switching node, presented in [58], [59]. The idea is based on the fact that network hardware is generally implemented as a pipeline through which packets flow and are processed in different stages of the pipeline. Thus, the API that enables the configuration of modular architecture and the transfer of packets from one component to the other is proposed. Components of the pipeline are modular and can be reused in other projects. For example, an IPv4 router was built using the reference pipeline, with five modules: (1) medium access control (MAC) layer reception queues, (2) input arbiter, (3) output port lookup, (4) output queues, and (5) MAC layer transmission queues. Other examples of network devices that are built using the reference design are 4-port network interface card (NIC), 4-port Ethernet learning switch, OpenFlow switch, etc.

OpenFlow switch implementations on NetFPGA-10G, ML605 and DE4 platforms, which have demonstrated the portability and flexibility of the proposed architecture, are presented in [60]. The switch design is described using the Bluespec System Verilog (BSV) language. Particular attention is devoted to solving the challenges of portability and flexibility through high modularity and configurability. For the design of the switch pipeline, the following modules were used: (1) flow table records composer, (2) flow table controller, (3) action processor, (4) arbiter i (5) switch controller interface. All modules are designed in such a way that machining functionality is independent of the platform (e.g., type of memory used, type of network or PCIe interface).

Network-attached FPGA concept - which enables the distribution of hardware design to multiple physical resources (e.g., FPGA devices), is presented in [61]. The proposed architecture, called OpenPipes, follows the basic principles of system design according to which the system consists of multiple modules: (1) processing modules, (2) flexible interconnections, and (3) a controller that configures interconnections, and manages the location and configuration of process modules. OpenFlow was used to implement interconnection architecture. Processing modules can be implemented in hardware or software and can be relocated while the system is operating, enabling real-time experimentation or migration from the old implementation platform to the new one.

Another example of network-attached FPGA concept is presented in [62]. A direct connection of FPGAs to a data center network (DCN) using integrated network cards is suggested. The FPGA is divided into three parts: (1) users logic, which implements custom applications, (2) network service layer which connects FPGA with DCN, and (3) management layer which performs resource management tasks. The integration of the proposed architecture into the cloud is envisaged using the new OpenStack service.

Network-on-chip enhanced FPGA - was used to develop a new programmable packet processor [63]. The new form of a packet processor, providing a high degree of flexibility and throughput of 400 and 800 Gbps, has been developed by interconnecting multiple protocol-specific processing modules. Instead of using the match tables that support the entire set of protocols, in the proposed design, packets are sent to the suitable modules depending on the protocol specified in the packet headers. Each processing module determines the actions that will be taken for that protocol, and which is the next processing module in the packet processing chain. Reconfigurable nature of the FPGA provides complete freedom in adapting and supplementing the processing modules and effectively brings programmability directly into the data plane.

2) SoC-BASED IMPLEMENTATIONS

In response to the code re-use problem in FPGA-based networking hardware, a new flexible legacy design support for SoC and system on FPGA (SoFPGA) platforms was proposed in [64]. The proposed architecture consists of the data plane and control plane bridges, in which the data plane bridge encapsulates the old design and integrates it into a new one.

Integration of NetFPGA-1G Output Port Lookup module into NetFPGA-10G design is given as an example. In this case, the data plane bridge is situated in the pipeline between the input arbiter and the output queues and is connected using the AMBA4 AXI-Stream interface. The control plane bridge is connected via the AMBA4 AXI interconnection interface to the data plane bridge. In this way, the control plane bridge has access to the internal data registers of the data plane.

OpenFlow switch implementation on SoC - is presented in [65]. The programmable platform ONetSwitch is based on the Xilinx Zynq-7045 SoC, which features a dual-core ARM Cortex-A9 Processor System and a Kintex-7 FPGA Programmable Logic (PL) within the same chip. One side of the Zynq PL is connected to four 1Gbps Ethernet and four 10Gbps SFP+ interfaces, and the other side is connected with Zynq processor system. The switch data plane is based on a hybrid software-hardware solution. The flow table lookup is performed in hardware, and in if there is no matching in hardware, the software is utilized. An algorithm, for flow table distribution into hardware and software, uses switching performance as an optimization criterion and is implemented within the hardware abstraction layer (HAL).

The hardware-software co-design of OpenFlow switch using SoC platforms is presented in [66]. The architecture of the proposed switch consists of a software agent and a hardware-based data plane. The agent is implemented as a Linux application running on an ARM processor and performs following tasks: sending packets to data plane, updating/deleting flow records, reading counters, and accepting packets from the data plane. The data plane is implemented as a modular pipeline on the programmable logic of SoC. Packets can enter the pipeline, consisting of the packet parser and lookup table, through the OpenFlow agent or physical interface. The link between the OpenFlow agent and the data plane is realized through the AMBA4 AXI-Stream interface, the packet transfer, and the AMBA4 AXI-Lite interface, for signaling.

3) NP-BASED IMPLEMENTATIONS

On the other hand, network processors have been used in a considerably smaller scope for implementation of SDN's data plane than reconfigurable hardware, but it is still important to review some of the most significant research.

ServerSwitch design, motivated by the cognizance that commodity Ethernet switching chips have become programmable, is proposed in [67]. The switching Ethernet chip is used for adaptive packet transfer while the server CPU is used to control and process traffic in the data plane. The prototype is implemented on the ServerSwitch card that uses the Broadcom switching chip, and is connected to the server via the PCIe 4x interface. Also, a software stack for card management and traffic control and processing in the data plane is implemented. The software stack contains the kernel component through which the card driver is implemented and the application component that provides the API to the driver.

ForCES router implementation - based on Intel IXP network processor is presented in [68]. Router architecture consists of CE and FE, where CE implements management and control planes, and FE implements management and data planes. The data plane is split into a fast and a slow path. Fast path is in charge of packet processing at line speed, while routing, network management and packet exceptions management are executed in a slow path. LFBs, that build up the data plane, are implemented using multiple microblocks in a flat plane. Microblocks differ from one vendor to another and perform the single function. The FE prototype is implemented on the Intel IXP network processor, where the fast path is directly mapped to the MicroEngine layer, while the slow path is running on the Intel XScale layer. Multiple FEs were interconnected using a switch based on the Advanced Telecommunications Computing Architecture (ATCA). The presented implementation of ForTER was used for the realization of LFBs defined in [35].

OpenFlow switch implementation - based on ATCA platform with architecture consisting of a data path (i.e., fast path) for packet forwarding and a control plane (i.e., slow path) for management and signaling, is presented in [69]. Data plane elements are implemented using the Broadcom Ethernet chipset on the AT8404 card that supports header parsing, packet classification, and frame forwarding by header field content. The packets which can not be directly forwarded are encapsulated and delegated to the embedded processor.

REMARK ON FLEXIBILITY AND PROGRAMMABILITY

Data plane implementations based on network processors have low flexibility from aspects of function placement and operation. The network processor programmability is generally limited to configuring the parameters of the data plane functions such as queue capacity, scheduling mechanism, packet header filter, etc. On the other hand, in the reviewed papers it has been shown that the use of reconfigurable hardware and suitable hardware description languages can achieve a high level of programmability, which positively affects the data plane flexibility. NetFPGA project, which stands out of all the reviewed approaches, simplified the process of implementing innovative network hardware using FPGA technology through the great support of the research community. A step further in the hardware programmability was made by SoC technology, which enabled the implementation of hybrid data plane architectures composed of a fast hardware path and a highly programmable software path.

B. SOFTWARE-BASED IMPLEMENTATIONS

Only after increasing the processing power of conventional computer systems, the software-based implementation of SDN's data plane became attractive both to researchers and industry. However, development of the idea about a software-based implementation of a programmable packet switching node, which later served as a useful tool for SDN's data plane implementation, began several years ago. By reviewing

the research dealing with software-based implementations, we have noted the following categories:

1) pure software-based implementations,
2) implementations based on virtualization techniques,
3) implementations supported by hardware-based acceleration.

1) PURE SOFTWARE-BASED IMPLEMENTATIONS

Click - a new software architecture for the realization of configurable and flexible routers is presented in [70]. Each Click router is built from the packet processing module, so-called elements. The elements are performing simple router functions such as queue processing, packet classification, and providing an interface to network devices. For a description of the router configuration, a declarative language that supports user-defined abstractions has been proposed. The declarative language, which is also readable to humans, is suitable for machine processing and can be easily translated into a directed graph, where elements are represented by nodes and a packet transmission lines between elements with branches.

Similar to the idea of Click, the extensible open router platform (XORP) is presented in [71]. Its design addresses four key objectives: (1) features, (2) extensibility, (3) performance i (4) robustness. XORP was conceived both as a stable platform and as a research tool that would allow smooth transfer of new ideas from the lab environment to the production network. It consists of two subsystems: (1) a high-level subsystem that performs routing protocols and other management processes in the user space, and (2) a low-level subsystem that manages data plane processes. Data plane is realized using the Click modular router, but it can also be implemented at the UNIX kernel level by exploiting the forwarding engine abstraction layer that abstracts the implementation-specific data to routing processes.

To increase the performance of the Click software router, the RouteBricks architecture, which enables parallel process execution on multiple processor cores within one or more servers, is proposed in [72]. The design is based on the Click Router extension with the support of allocating specific elements of the Click router to particular processor cores. To achieve this, the 10G network adapter driver is additionally extended with support for multiple queues and support for NIC-driven batching. This ensured that one packet is processed on only one processor core, and the number of input/output (I/O) transactions is also reduced. By implementing a prototype RouteBricks router, named RB4, made up of four servers connected by 10Gbps links in a full-mesh topology, it has been shown that a bandwidth of up to 40Gbps can be achieved.

OpenFlow switch implementation - based on Click modular platform is described in [73]. To create a hybrid model that allows packet- and flow-based processing, an OpenFlowClick element has been added within the Click router, that enables rule tables management through the OpenFlow protocol. OpenFlowClick runs as a Click kernel module, and uses the *secchan* and *dpctl* tools to communicate with the OpenFlow controller. Within the OpenFlowClick element, the data path module performs a rule checking and packet forwarding, and the control plane module manages the forwarding rules table according to the controller commands. Linear and hash tables are used for the implementation of wildcard and exact matching.

Another software-based implementation of the OpenFlow switch is presented in [74]. The switch architecture is based on the NetBee library and consists of: (1) ports that use (2) NetBee Link components for switch and network interfaces connection, (3) NetPDL [75] description of OpenFlow 1.3 protocol formatted in XML, (4) NetBee XML protocol description parser, (5) flow table, (6) rules grouping table, (7) meter table, (8) oflib library for OpenFlow messages conversion to internal format and vice versa, and (9) secure channel to the OpenFlow controller.

A cost-effective alternative to SDN implementation using Raspberry Pi single-board computers and the OVS software switch is proposed in [76]. Although the new architectural aspects of the data plane SDN are not presented in this paper, it has been shown that single-board computers can be used as a platform for execution of software switches. The performance of the prototype implemented is similar to that achieved by using a hardware switch based on the NetFPGA-1G platform.

In addition to the implementation of OpenFlow switches that are published in research papers, here we list some other open-source implementations of OpenFlow switches and supporting libraries:

- OpenFlow v1.0 reference implementation [77] - written in C,
- Indigo [78] - support for physical and hypervisor-based switches written in C,
- Pantou [79] - port of OpenFlow implementation for OpenWRT wireless platform written in C,
- OpenFaucet [80] - implementation of v1.0 written in Python,
- OpenFlow Java [81] - OpenFlow stack written in Java,
- oflib-node [82] - implementation of v1.0 and v1.1 protocols in the form of libraries for Node.js.

Combination of emulation and simulation - to support the realization of large network experiments is proposed in [83]. The proposed architecture, inspired by the idea of SDN, separates the control and data planes so that the control plane is emulated and the data plane is simulated. Within the simulated data plane, there are common elements of a network simulator based on discrete-event simulations (DES):

- event queues,
- queue scheduler,
- event processor,
- network status and statistics register,
- topology containing a simulated logic of network nodes.

The following events are held in the queues: (1) start of an application, (2) flow arrival, (3) flow departure, (4) arrival of message from a control plane, and (5) departure of message to the control plane. The flows consist of aggregated packets

with common headers. The flows are separated into incoming and outgoing, to take into account the traffic losses. In a presented example of the SDN implementation, switches and end nodes are implementing the OpenFlow protocol within DES, while controllers are running as real and independent software instances.

2) IMPLEMENTATIONS BASED ON VIRTUALIZATION TECHNIQUES

Machine virtualization technique - has been used for the first implementation of virtualized data plane, called Mininet [84]. Mininet is the networking virtualization environment based on Linux virtual machines running on standard platforms such as VMware, Xen, and VirtualBox. It allows the creation of virtual networks by setting up and interconnecting the host processes within the network namespace. For host interconnection virtual Ethernet (*veth*) is used. The user can use various building elements for implementation of the SDN-based virtual network: (1) links made as *veth* pairs, (2) hosts realized as shell processes, (3) switches implemented as OpenFlow software switches, and (4) controllers that can be run anywhere in a real or simulated network.

The idea of scaling the Click modular router performance by increasing the number of instances of Click routers running within miniature virtual machines is presented in [85]. These miniature virtual machines are called ClickOS, and are running on a Xen hypervisor. Xen hypervisor has a shared network *netback* driver, which communicates with hardware on the one side, and through shared memory with the ClickOS *netfront* driver on the other side. The task of the *netback* driver is to forward packets between from network adapter to shared memory, and vice versa, over a virtual network interface. On the other hand, the *netfront* driver is scheduling packets from shared memory to the transceiving interfaces of Click router (FromClickOS and ToClickOS), and vice versa. This builds a bridge between the Click router and NICs while preserving all gains of virtualization.

Network/NIC virtualization technique - can be used for implementation of generic high throughput bus or in a concrete case for connecting virtual machines. In that sense, a virtual local Ethernet (VALE) is proposed in [86]. VALE, using the *netmap* API [86] as a communication mechanism, exposes ports to hypervisors and processes. The core *netmap* is based on shared memory, which represents the interface between network hardware and packet processing applications. Within that memory, packet transmission and reception buffers are assigned to each network interface, and two circular arrays called *netmap* rings for storage of metadata about transmission and reception buffers.

The software switch mSwitch [87] has simultaneously responded to several shortcomings noted in the previous solutions by utilization of techniques for network interfaces virtualization and the separation of switching from packet processing. mSwitch provides: (1) flexible data plane, (2) efficient processor utilization, (3) high throughput, (4) high packet processing intensity, and (5) high port density. The central principle of the proposed architecture is the division of the data plane into the switch fabric which is responsible for packet switching and the switch logic, which is the modular part of the switch, responsible for the packet processing. This allowed a high throughput, while maintaining the high level of programmability of the packet processing functions.

Open vSwitch (OVS), a multilayer virtual switch presented in [88], is intended for networking in virtual production environments and supports most hypervisor platforms. In the OVS architecture, the packet forwarding is performed using two components:

1) *ovs-vswitchd* daemon in the user space which is identical for all operating systems,
2) high-performance datapath kernel module written for the target operating system.

The datapath kernel module is responsible for receiving packets from a NIC or a virtual machine, and its processing according to the instructions given by the *ovs-vswitchd* module. In the case that there are no defined processing rules for the specific packet in the kernel module, this packet is forwarded to the *ovs-vswitchd* module, which then makes the decision on further processing and returns it together with the packet. When the OVS is used as an SDN switch, then the agent side of the OpenFlow protocols is running inside the *ovs-vswitchd* module.

3) IMPLEMENTATIONS SUPPORTED BY HARDWARE-BASED ACCELERATION

The problem of the limited performance of the pure software-based implementation of packet switching nodes has been addressed in some research by utilization of hardware-based acceleration.

Graphics processing unit (GPU) based acceleration framework - has been used for development of a software router PacketShader as presented in [89]. The challenge of maintaining high forwarding rate while preserving sufficient processing power for different routing applications is solved as follows: (1) the I/O engine for fast and efficient packet processing is implemented, (2) routing table lookup and IPsec encryption are offloaded from the main processor to GPUs. I/O engine functions, which are implemented at the kernel level, are used for kernel-level packet handling operations. The remaining packet processing operations are executing in the multi-threading application in user space with the help of three callback functions: (1) pre-shader, (2) shader, and (3) post-shader. The pre-shader function performs fetching of packet parts from the receiving queues to the GPU. The shader function performs processing of the packet within the GPU kernel, and the post-shader function delivers processed parts of the packet to the destination ports. In this way, an efficient pipeline for packet processing by FIFO principles has been established. By implementing the prototype of the router, it has been shown that high throughput such as 40Gbps can be achieved.

NIC-accelerated Ethernet switch implementation - called CuckooSwitch [90], combines the hash-based forwarding information base (FIB) design with Intel's Data Plane Development Kit (DPDK) [91] platform that performs inbound/outbound packet transmission between hardware and user-space threads. The packet processing pipeline has three stages: (1) reception of the packet via the NIC and its storage in the reception queue using direct memory access, (2) processing of packets from queues using a worker thread in the user space, and (3) scheduling of processed packets into output queues associated with appropriate output ports. The number of input queues has been chosen to correspond to the number of worker threads, where each worker thread is allocated to one processor core. In this way, competition and synchronization overhead have been avoided. In the second stage of the pipeline, the FIB search is performed based on the destination MAC address of the Ethernet frame. FIB supports dynamic updating of rules in real time and reading of records from multiple concurrent worker threads, eliminating the need for storage of numerous copies of FIB content. By implementing a prototype, in [90] it has been demonstrated that using a standard server with eight 10Gbps Ethernet interfaces, processing power of 92.22 Mpps can be achieved for 64B packets.

NIC-accelerated OpenFlow switch implementation - based on the Intel DPDK library is presented in [92]. By using the DPDK library, implementation costs have been reduced in terms of the packet I/O overheads, DRAM buffering, interrupt processing, kernel structure overheads, and copying data when changing the context from the kernel space to the user space and vice versa. DPDK enables, through Direct I/O mechanism, direct data transfer between the program running in user space and input/output cards.

REMARK ON FLEXIBILITY AND PROGRAMMABILITY

Pure software-based data plane implementations are characterized by excellent flexibility due to high level of programmability and configurability of forwarding functions. For example, software architecture of the modular router Click allows the realization of arbitrary data plane structures using fully programmable packet processing modules. In the reviewed papers it has been shown that high flexibility, in terms of the granularity of dynamic resource management, may be achieved by parallel execution of instances of software implementations on multi-core processors. Virtualization-based techniques additional contribute to flexibility, as they enable efficient scaling of forwarding functions. The use of hardware-based acceleration enables the increase of performance of software-based implementations without losing inherent flexibility. A large number of software-based implementations of the OpenFlow switch indicates that the software-based approach to the realization of the SDN's data plane is powerful, especially if their application is planned in modern data centers which often use virtualization techniques.

V. IMPLICATIONS OF SDN-RELATED RESEARCH ON DATA PLANE EVOLUTION

The previous sections focused on standard SDN's data plane architecture, such as ForCES and OpenFlow, and their implementation using software and hardware technologies. On the other hand, an overview of SDN-related research whose results have implied the data plane evolution, is given in this section. Under the data plane evolution, we indicate a gradual deviation from the original data plane architectures given with ForCES and OpenFlow specifications, resulting in the need to address the problem of programmability and flexibility of the data plane in a more generic way. By reviewing these studies, we have found that the treated problems can be classified into the following categories:

1) performance,
2) energy consumption,
3) quality of service (QoS),
4) measurement and monitoring,
5) security and reliability,
6) support for various network technologies,
7) network and network functions virtualization.

Therefore, this section is organized in accordance with the above mentioned categories. Within each category, reviewed research is organized according to the problem-solving approach. Given that many problem-solving approaches are common to several categories of problems, here is given a brief overview of identified problem-solving approaches.

Data plane programming - implies the introduction of programmed packet processing into the data plane. The machine structure, which executes program instructions on packets, is mainly fixed and predefined.

Stateful packet processing - allows packet processing which is aware of the state of the data plane. A lot of research examined in this section has shown that the stateless nature of the OpenFlow switch does not provide adequate support for processing of packets coming from a stateful protocol (e.g., TCP, FTP) or the implementation of some mechanisms such as a stateful firewall. Stateful packet processing is most often achieved by introducing finite automata into the data plane.

Reconfigurable architectures - most commonly based on reconfigurable technology (e.g., FPGA), allow the implementation of a variable data plane structure. That has been used in some research to solve a specific problem from the above categories by increasing the flexibility of the data plane.

Physical layer management - is a frequently used technique for managing the energy consumption of the network device's physical interface.

New structures of flow tables - are introducing, in addition to basic information such as packet headers of a specific flow, additional data to support the treatment of issues related to data plane performances, energy consumption, QoS, etc.

New mechanisms for flow tables lookup - often accompany structural changes of flow tables. In some cases, new mechanisms are based on the enhancement of existing OpenFlow flow table lookup mechanisms.

New packet classification mechanisms - allow packet classification based on the header of higher-layer protocols. In this way, they enable the implementation of advanced mechanisms such as DPI.

Hybrid architectures - are data plane architectures implemented using a combination of hardware and software technologies or using different types of hardware technologies (e.g., FPGA and CPU). Such architectures enable efficient distribution of packet processing tasks according to the affinities of the particular technology.

New data plane architectures - are proposed in a significant number of research as an answer to the limited flexibility of OpenFlow and ForCES data plane architectures.

Hardware abstraction layer (HAL) - is a method for providing support for non-IP networking technologies in such a way that specificities of target technology are abstracted to the data plane through suitable processes and interfaces independent of underlying network technology.

At the end of the section, we have established a correlation between treated problems and problem-solving approaches, which is the first step towards the generalization of approaches to increase the programmability and flexibility of the SDN's data plane.

A. PERFORMANCE

The performance improvement problem of SDN has been addressed by introducing various changes to the data plane. The focus of [93]–[103] was on changes in the flow table structure and the introduction of new flow table lookup or packet classification mechanisms. On the other hand, in [104]–[115], the focus was on changing data plane architecture by using hybrid software-hardware architectures or combinations of different types of hardware, by introducing reconfigurability into the data plane, and by introducing stateful packet processing,

New structure of flow table - which supports load balancing based on regular expressions, is proposed in [94]. Half-SRAM was used on NetFPGA because in load balancing scenarios, it is necessary to keep a large number of records in flow tables. The hardware plane takes care of the longest prefix matching (LPM) in the dFA structure, and the software plane manages the data structure by inserting/removing the rules in/from the forwarding table.

The implementation of the packet switch with Bloom-filter based forwarding, called zFilter, is described in [93]. The proposed forwarding mechanism is based on the identification of links instead of nodes. Packet switching nodes do not need to maintain any status except link identifier information for each interface. The forwarding information is constructed based on the aforementioned link identifiers and is transmitted in the header of the packet as the Bloom filter structure. Based on the presence of the link identifier in the Bloom filter structure carried in the packet, each packet switching node decides to which interface a packet should be forwarded.

The new method of the hardware-based organization of the forwarding tables in SDN switches is presented in [103]. Given the advantages of parallel processing on the FPGA chip compared to serial processing on network processors or general purpose processors, the proposed solution uses all 512 bits of the header in a wildcard-based lookup.

New mechanism for flow table lookup - called DevoFlow, which reduces the number of interactions on the switch-to-controller interface and the number of records in ternary content-addressable memory (TCAM), through the aggressive use of wildcard rules, is introduced in [95]. The proposed modification is based on the introduction of two new mechanisms for devolving the control from the controller to the switch: rules cloning and local actions. Rules cloning allows the creation of new rules for micro-flows based on templates, which rule search diminishes to direct matching, thus reducing the use of TCAM. Local actions allow prediction of rules and their establishment without contacting the controller. Another contribution of DevoFlow is lessening the need for statistics transmission for less dynamic flows through the use of three mechanisms for efficient statistics collection: (1) sampling, (2) triggering, and (3) approximate counting.

The classic routing table lookup is modeled as the problem of the LPM and is divided into three main categories:

1) TCAM-based solutions which provide a deterministic and quick lookup,
2) hash-based solutions which provide a quick lookup with simple table update mechanisms,
3) trie-based solutions.

Instead of optimizing classical models, the brand new model, the Split Routing Lookup Model, is proposed in [102]. The basic idea is to divide the original flow table into two smaller, perform LPM lookup over them, and to aggregate two results into one. This results in savings in chip resources, and increased performance by introducing parallelism in the lookup process.

KeyFlow, proposed in [98], is a new approach to building a flexible network-fabric based model. The flow table lookup mechanism in the forwarding engine was replaced by simple operations based on the residual number system. Principally, the proposed model is based on source routing. Unlike Multiprotocol Label Switching (MPLS) based solutions, where there is a need for intensive communication between the core and the controller to establish end-to-end connections, all possible routes are ready for use in KeyFlow and only the appropriate route identifier for the ingress packet has to be allocated. In this way, round-trip time reduction was achieved by more than 50 percent.

The caching system for the SDN based on the wildcard rules, called *CAching in Buckets* (CAB), is presented in [101]. The basic idea of the CAB is the partitioning of the field into logical structures, called *bucket*, and bucket caching along with the associated rules. The CAB switch is implemented as a two-stage pipeline consisting of bucket filters and flow tables. In the first stage, the matching of the packets is done in all buckets, and in the second stage, the flow table lookup is performed. Packets that do not belong to a single bucket are

encapsulated and sent to the controller. Logically, the bucket filter and the flow table are two tables that can be implemented using two separate TCAM memory or one multiple-lookup TCAM memory. The CAB solves the problem of the rules dependence and achieves efficient utilization of network bandwidth by reducing communication between the switch and the controller.

New packet classification mechanism - based on recursive flow classification is presented in [96]. Extended recursive flow classification (ExRFC), using the combination of SRAM and TCAM memory, improves the parallelization of the classification processes and exploits the hardware resources of most hardware platforms better.

The 2-dimensional pipeline for packet classification on the FPGA, which consists of multiple self-configurable processing elements (PE), is presented in [97]. The proposed architecture achieves scalability, regarding the size of the rule set, while maintaining a high-throughput packet classification in the OpenFlow switch. The modular PE can perform matching by the scope and by the prefix, making it suitable for different types of packet headers. Connecting PEs in the 2-dimensional pipeline is possible in two directions:

1) horizontal propagation of the bit vectors of the PE output registers in a pipeline fashion,
2) vertical propagation of the packet header bits (PE input registers) in a pipeline fashion.

A complex packet classification operations can be realized, by using PEs striding and clustering problem-solving techniques, whereby the clock signal frequency is not limited by the length of the packet header and the size of the rule set.

The new hardware solution for the configurable packet classification is presented in [99], [100]. The performance of different classification algorithms was analyzed through two approaches: (1) a multi-field lookup, and (2) a single-field lookup. It has been shown that a parallel combination of different lookup algorithms based on one packet header field has achieved better performance than using a lookup algorithm based on multiple packet header fields. Therefore, the design of a hardware-based classifier architecture is proposed, which achieves optimal lookup performance by running the best set of algorithms for a given type of record in the flow table. Sharing memory resources between different lookup algorithms has resulted in efficient memory utilization. The SDN controller selects the best combination of search algorithms, and following the decision, configures the appropriate memory blocks on the hardware platform of the switch. The packet classification process is performed within a four-stage pipeline:

1) splitting of packet header into multiple fields over which individual lookup algorithms will be performed,
2) parallel execution of lookups, using a selected set of algorithms,
3) combining lookup results into a single tag with the highest priority,
4) reading the highest priority match rule based on the tag from the preceding stage.

The proposed architecture can follow the evolution of SDN applications by the simple extension of the existing set of lookup algorithms with a new one.

Hybrid architecture - consisting of hardware and software components, was used in [104] for the implementation of the uniform resource locator (URL) extraction mechanism from the hypertext transfer protocol (HTTP). The hardware component is an extension of the IPv4 reference router on the NetFPGA platform, and is implemented by modifying the *Output Port Lookup* module in the reference design. Its task is to extract the HTTP GET request and send it to the software component. The software component extracts the URL from the HTTP GET request and updates the internal database. This paper presents an approach to the performance improvement of the deep packet inspection by applying the hybrid architecture of the packet switching node.

The OpenFlow switch reference design, accelerated using multi-core network processors, is presented in [105]. In the proposed design, 16 micro-machines (programmable cores) of the network processor are programmed to perform tasks of receiving packets, sending packets, processing packets, managing queues and serving orders, communicating via PCI bus, etc. The implementation of the given design consists of the software on the host and the network processor (NP) acceleration card. On the host side, the OpenFlow software communicates with the NP accelerator card via the PCIe bus using the kernel module. The experiments carried out showed a reduction in packet delay by 20% compared to the conventional OpenFlow switch design.

The sNICh architecture, which is a combination of the NIC and a data center switching accelerator, is proposed in [106]. sNICh uses acceleration hardware in the form of a PCIe card, for offloading the server concerning the packet forwarding. The data and control planes in the sNICh architecture are separate, where the data plane is implemented in the NIC hardware, and the control plane is implemented within the sNICh backend. In addition to standard NIC functionality, sNICh also realizes a flow-based switch whose flow table is implemented using TCAM. In the data plane, a copy engine and memory-to-memory direct memory access engine for direct access to shared system memory are implemented, to improve performance in communication between virtual machines (VM) within the data center.

Inspired by the idea of the packet processing offloading from the processor to the NIC, the architectural design that improves the flow table lookup performance on a PC-based OpenFlow switch is proposed in [107], [108]. The proposed solution is based on flow caching and placing the lookup process in the fast path of the switch. The acceleration of OpenFlow flow table lookup process is achieved by using the classification features of Intel 82599 10GbE controller found on modern 10GbE NICs.

The use of the CPU in switches to handle not only control traffic but data plane traffic as well is presented in [110]. A powerful processor has been added to a commodity switch and connected by a high-bandwidth link to the

application-specific integrated circuit (ASIC), which can be programmed to redirect a portion of the traffic to the processor. In this way, the design limitations of the switches, regarding the forwarding table size and the depth of the packet buffers, have been overcome. Additionally, adding a CPU to the network device has increased the data plane programmability.

CacheFlow, a new splicing technique of a large number of unpopular forwarding rules, from dependency chains, to a small number of new rules, aiming for the cache pollution avoidance, is presented in [111]. It combines high-throughput hardware switches with large flow tables of the software-based switches. Hardware switches implement cache on TCAM, and in the case of cache miss, software switches are engaged, thus avoiding unnecessary and time-consuming communication with the controller.

Following the trend of adding powerful processors to conventional switches, in [114] is argued that a hybrid software-hardware switch can reduce the time needed to install rules in the flow table. Accordingly, ShadowSwitch (sWs) was proposed as the prototype of the OpenFlow switch which implements this design. Besides the hardware switch (HwSw), a high-performance software layer (SwSw), which performs packet forwarding, was introduced in the fast path of sWs architecture. The control logic (sSwLogic), whose task is to manage the record installation in the flow table, is placed in a slow path of the switch. The prototype is implemented using the commodity hardware-based OpenFlow switch and the OVS instance which runs on the server.

Reconfigurable architecture - based on a virtual emulated network on a chip, called Diorama network, is proposed in [109]. The prototype is implemented on the dynamic reconfigurable processor DAPDNA-2, which consists of a high-performance digital application processor based on reduced instruction set computer (RISC) architecture and distributed network architecture (DNA). DNA connects 376 PEs and is used to construct emulated nets on the chip, where each PE emulates different functions of the actual router or link. Test results have shown that the prototype can perform the shortest path calculation 19 times faster than conventional solutions.

FlexForward, which enables flexible adaptation of forwarding algorithms in software-defined DCN, is proposed in [112]. Reconfiguration of forwarding mechanism on switches is supported by the OpenFlow protocol and implemented by OVS extension. Forwarding mechanisms are implemented between flow extraction and flow table lookup processes. Performance improvements have been achieved by introducing an additional feature to skip the flow table lookup process. Supported forwarding mechanisms are:

- regular OpenFlow - used when FlexForward is switched on, as long as the OpenFlow switch does not get command to change the forwarding mechanism,
- hypecube topology [116] - used in the server-centric DCN,
- KeyFlow - used in arbitrary topology.

Stateful packet processing - enables offloading of simple networking processes, such as filtering and counting, from the control plane to the network switches. By using the open packet processor (OPP) in [115], a stateful DPI application was implemented as a pipeline consisting of two tables: one for selecting an output interface and the other for implementing DPI functionality. OPP will be described in more detail in the Section VI.

The use of Paxos protocols for distributed and programmable network systems to improve SDN performance is demonstrated in [113]. For Paxos protocol implementation, it was necessary to add support for the roles of coordinator and acceptor in the OpenFlow switch logic. According to the Paxos protocol, the coordinator must support the generation of a unique round number and a monotonically incrementing sequential number. The acceptor switch must support the storage and stateful comparison of the received random number with the appropriate field in the packet header, and maintenance of the local state by the protocol. It is proposed to implement these functionalities at the edge of the network using programmable NICs.

B. ENERGY CONSUMPTION

The issue of energy efficiency and energy consumption in the SDN has also been addressed in several ways. Some have focused on reducing energy consumption in memory for storing flow table and introducing new flow table lookup mechanisms. Others have decided to reduce the energy consumption of network's physical layer by introducing the support for physical layer management and by abstracting underlying hardware.

New structure of flow table - which stores flow identifiers (Flow-ID), represented by a smaller number of bits, instead of standard flow records, is proposed in [117]. The proposed Compact TCAM summarizes the information about the flow to the size needed to identify unique flows. It has been experimentally demonstrated that it is possible to save energy by about 2.5 times the standard layer 2 switches, and up to 80% compared to OpenFlow switches.

New flow table lookup mechanism - based on flow tagging, called Tag-In-Tag, is proposed in [118]. Tag-In-Tag is based on a generally-perceived phenomenon in networks:

- flows travel by paths,
- paths form a deterministic set, i.e., all source and destination pairs are known in advance,
- multiple flows can travel along the same path.

Based on that, the proposed mechanism denotes a packet with two tags: PATH TAG used for packet routing, and FLOW TAG that associates the packet with the corresponding flow. PATH TAG encapsulates within FLOW TAG. Within the switch, the data path is divided into two TCAMs, one for the incoming and the other for the outgoing packets. On edge switches, TCAM for incoming packets contains full headers, and TCAM for outgoing packets holds only PATH TAG and FLOW TAG. In core switches there is only TCAM that contains PATH TAG and FLOW TAG. In the end, the lookup

is performed over a small number of bits in parallel, thereby achieving a TCAM energy savings of up to 80%.

Motivated by the limitations of commercial switches at the time, regarding latency and energy consumption, improvements that simultaneously reduce energy consumption and switch latency, while retaining the flexibility offered by the OpenFlow, are proposed in [119]. The proposed enhancements exploit local time stationarity of network traffic and predict the affiliation of the packet to the flow for each port of the switch. Thus, they try to avoid traditional flow-based or TCAM lookups, which simultaneously reduces both energy consumption and latency of forwarding.

Physical layer management - as an extension for OpenFlow switch with support for different power saving modes is presented in [120]. It includes the definition of new messages within the OpenFlow standard and the design of a separate controller that manages port powering. The prototype is implemented by combining a NetFPGA platform and a specially designed hardware controller.

Hardware abstraction layer - for a unified and straightforward representation of power management features in heterogeneous data plane hardware, called Green Abstraction Layer (GAL), is presented in [121]. On the side of network devices, manufacturers can implement power management primitives (PMP) by introducing specific hardware-level elements such as controllable clock generators, manageable power supplies, sleeping transistors, etc. On the network management side, the GAL abstracts the PMPs and enables simple power management through higher layer protocols.

C. QUALITY OF SERVICE

New structure of flow table - supporting fine-grained tracking of flows with a separate flow classification, is proposed in [122]. The flow table is divided into three different tables: the flow state table, the forwarding rules table, and the QoS rules table. The forwarding and QoS information are searched in the rule tables and linked to the record from the flow state table on the arrival of the first packet from a new flow. The arrival of the next packet from the same flow only requires a referral to the flow state table without the need to search through the forwarding rules table and the QoS rules table again. To assure performance at the micro-flows and the aggregate flows levels, a flow-based packet scheduling algorithm was developed. The proposed architecture was implemented using the Cavium OCTEON CN5650 multi-core processor, whose 12 cores were assigned as follows: one core for the OpenFlow agent, eight cores for flow-based packet processing, one core for future functionalities, one core for QoS coprocessing, and last core for the server. All cores share data from the forwarding rules, QoS rules and flow state tables, and queues.

New data plane architecture - is introduced in [123] to support QoS experimentation in OpenFlow testbed Ofelia [124]. The proposed QoS framework did not restrict the configuration of queues regarding minimum and maximum speed, as was the case with OpenFlow. Configuration of queues is provided through additional control protocols such as OF-Config [125] and NETCONF [126].

B4, introduced in [127], is Google's approach to SDN-based DCN implementation with QoS support. The proposed SDN architecture is based on OpenFlow, but due to the limitation of existing switch architecture concerning low-level behavior management, a custom hardware platform was used. B4 switches consist of more commodity switching chips organized in two-stage Clos topology. Each chip in the first stage (input stage) of the Clos topology is configured to forward incoming packets to a second stage (backbone) except when the packet terminates on the same chip. Chips from the second stage pass the packets to the chip from the first stage depending on the packet destination. The specially developed OpenFlow agent running inside the B4 switches mediates in the configuration of the forwarding table in this non-standard pipeline by translating the OpenFlow messages into the corresponding chipset driver commands.

Since the OpenFlow switches implemented in the OVS did not support the queue configuration at that time, the QueuePusher architecture based on the OVSDB standard supported in the OVS was designed in [128]. Since OVSDB is not part of the existing OpenFlow controllers, QueuePusher has been created as an extension of the existing Floodlight controller interface to simplify the procedure for creating queues within the OpenFlow switch. Although there were no direct changes to the SDN's data plane, it is remarked that there was an OpenFlow protocol constraint regarding the configuration of the queues which belong to the SDN's data plane.

The API for configuration of priority queues on switch ports, based on an extension of the interface between the SDN controller and the switch with OVSDB protocol support, is proposed in [129]. Unlike the solution proposed in [128], the QoS abstraction model of OVS switch is defined here. The model is stored within the OVS database and accessed through the OVSDB protocol. A QoS object which specifies the maximum speed that can be shared between the priority queues is defined on each OVS switch port. The QoS configuration module inside the controller does not keep information about the state of switch queues, but only maps the QoS configuration to the switch port. Retaining information about the state of queues within switches makes it easier to maintain data consistency.

Hybrid architecture - based on the combination of FPGAs and commodity switching hardware, is proposed in [130]. Extending SDN flexibility by making queuing and scheduling decisions inside a fast path of the switch is achieved by adding small FPGAs, with well-defined interfaces to packet queues on the switch, in a fast path of the switch. As proof of the proposed concept, two scheduling schemes have been described and implemented: CoDel and RED.

With the addition of the FPGA controller used for the deterministic guaranteed-rate (GR) services, the relocation of the network intelligence from the control plane to the data plane is presented in [131]. The data plane within the developed solution consists of eight FPGA controllers and

packet switches of minimal complexity implemented on the Altera Cyclone IV FPGA. The FPGA controller allows the provision of deterministic GR services via IP routers, MPLS, Ethernet, InfiniBand and Fiber Channel switches. By using optoelectronic packet switches implemented on a single chip, it is possible to achieve an aggregated capacity of 100 Tbps.

Stateful packet processing - was used in [132] to create the autonomous QoS management mechanism in the SDN (AQSDN). AQSDN enables the independent configuration of QoS features in the switch using the OpenFlow and OF-Config protocols. Autonomy is ensured by implementing the packet context-aware QoS (PCaQoS) model inside the data plane, which enables the switch to know the context of the packet and to respond locally. The PCaQoS model consists of two components: packet context-aware packet marker (PCaPM) and packet context-aware queue management (PCaQM). PCaPM is based on a multi-color DSCP-based packet marking, while PCaQM coordinates these activities at the integrated flow level. The system prototype is implemented by extending the software switch Ofsoftswitch13 [74].

Hardware abstraction layer - for representation of physical network layer in the hierarchically autonomous QoS model, is proposed in [133]. In the proposed model, a data plane is precisely cut into multiple virtual networks with the capabilities of the dynamic allocation of resources. For the implementation of the proposed architecture, the following mechanisms have been used:

- redistribution of virtual resources based on the context,
- network virtualization,
- autonomous network structure.

Redistribution of virtual resources is achieved by cutting overall physical network infrastructure resources according to the needs of a single service or application. Network virtualization is enabled by introducing a flow structure with an extendable packet matching mechanism. Autonomous network infrastructure is ensured by introducing control loops in the control plane which are responsible for the settings configuration on physical network elements.

D. MEASUREMENT AND MONITORING

New structure of flow table - to support sampling of packets covered by wildcard flow records in OpenFlow switch, is proposed in [134]. In the mentioned paper is was noted that traffic intensity between controllers and switches can be significant in networks such as DCN, which may result in slower forwarding of new flows. One way to reduce control traffic is the usage of wildcard records for the creation of default routes in the network. Since switches do not keep track of the flows covered by wildcard records, the controller has no more information about individual flows.

New data plane architecture - which, in addition to data and control planes, introduces a history plane to support the packet history logging regarding its entire journey through the network is presented in [135]. The proposed history plane includes NetSight servers and the coordinator. In addition to basic SDN tasks, controllers are responsible for configuring packet history filters (PHF) on switches. PHFs are described in a language based on regular expressions, and specify the path, switch state, and packet header fields for the history of the packet of interest. When the packet passes through the switch, a postcard is generated based on the PHF trigger and delivered to the NetSight server. A postcard represents a packet summary that contains elements essential to tracking its journey through the network. NetSight servers collect postcards, process them and store them in compressed lists, and on the request submit compressed packet history to the coordinator. The coordinator is responsible for processing the history of the packet and generating useful information for the network monitoring applications.

Concerning the evolution of OpenFlow-based SDN founded on the principles of network devices generality, distributed control plane and simple packet processing, in [136] they argue that three principles need to be satisfied to maintain vertical scalability:

1) control functionalities should remain in the control plane domain, other than those which promote the efficiency of packet processing and adapt to the hardware and software requirements of the data plane,
2) control functionalities cannot change the basic data plane processes,
3) the collection of statistics in the data plane should not affect the accuracy and validity of the measurement and should not cause an increase in the control plane load.

Following these principles, it is proposed to offload the control plane from the control messages, by introducing a statistical server that closely cooperates with network devices in the data plane and submits collected statistics only at the request of the controller.

Stateful packet processing - was used in StreaMon [137] to separate the program logic of the traffic analysis applications from elementary primitives implemented in the network device probes. StreaMon abstracts the measurement process through three phases:

1) identification of the entity being monitored,
2) measurement by applying efficiently-implemented primitives to the configurable fields of the packet header,
3) making decisions using extended finite-state machines (XFSM).

The implementation of this abstraction is enabled by using a stream processing engine consisting of four layers: (1) a layer of events that parse the recorded packets, (2) a layer of metrics applied to parsed packets, (3) a layer of features in which different statistics are derived from the calculated metrics, and (4) a layer of decision in which the measurement application logic is executed.

Hybrid architecture - which enables the introduction of software-defined counters by connecting ASIC to the general purpose processor and the cost-effective DRAM, and replacing the classic counters with small buffers, is proposed in [138]. The principle of work is the following. With the

arrival of the packet, instead of incrementing the counter, ASIC creates a record of that event and adds it to the buffer. The buffer is divided into several blocks which, when filled, are transmitted to a general purpose processor. A general purpose processor based on the block content is updating the counters located in the DRAM.

Similar to previous research, the software-defined measurement architecture, OpenSketch, which separates data plane measurement from the control plane, is proposed in [139]. In data plane, OpenSketch provides a simple three-stage pipeline: hashing, filtering, and counting. A measurement library has been created, which enables the automatic configuration of the data plane pipeline and the switch memory allocation for each measurement task. The prototype is implemented on the NetFPGA platform, by inserting packet header parsing, hashing and lookup modules, and SRAM-based counters in the reference switch pipeline.

Data plane programming - as a solution for network monitoring is proposed in [140]. Within the proposed solution, there is a simple programmable interface which allows end nodes in the network to query and calculate over the switch memory using tiny packet programs (TPP). The TPPs are embedded in the packet headers and contain several instructions for reading, writing or performing arithmetic operations over SRAM data or processor registers. The pipeline of the proposed solution is based on an ASIC containing a TPP processor (TCPU) between L2/L3/TCAM tables and the memory of the output queues. TCPU is based on a RISC processor, which executes instructions in five stages:

1) fetching,
2) decoding,
3) executign,
4) reading from memory,
5) writing to memory.

Examples of supported statistics which can be obtained from the switch memory are: counters associated with L2/L3 flow tables, link utilization, number of received/sent/dropped bytes, queue size, etc.

E. SECURITY AND RELIABILITY

Although the security [141]–[144] and reliability [145]–[148] issues of the network are of different categories, they are commonly concerned with detection, isolation, and rapid resolution of problems that can degrade the performance or completely impair normal network operation. The SDN's data plane is attractive as a place where security and reliability issues are solved because it is the first target for potential threats to security or network reliability.

Despite the fact that there was no change in the switch architecture, in [142] it is shown that by relocating relatively straightforward access control operations to a data plane, the load of the SDN controller can be reduced, thereby increasing the scalability and security of the entire network.

New data plane architecture - which enables fast detection of the impairments along the entire path, is proposed in [145]. It is shown that the data plane recovery can be achieved in less than 50 ms by the relocation of the connection monitoring from the control plane to the data plane. To do this, generators of monitoring messages, whose absence on the destination switch side can point to problems in the connection, were added to the OpenFlow switch architecture.

New packet classification mechanism - as an OpenFlow extension to support detection and blocking of the distributed denial-of-service (DDoS) attack through content-oriented networking, is introduced in [141]. In proposed extension an OpenFlow switch can respond to the requested content based on URL in requests. To achieve this, interceptor of the packet with the appropriate URL and the rate limiter were added to the hardware architecture of the OpenFlow switch.

Hybrid architecture - for packet classification, called HyPaFilter, is proposed in [143]. HyPaFilter exploits the advantages of parallel and massive hardware-based packet processing and the large inspection capabilities of software-based packet filters. It partitions user-defined policies for packet processing into simple parts executed on specialized hardware and complex parts run in the software. The firewall prototype is implemented by a combination of the NetFPGA-SUME hardware platform and *netfilter/iptables* software on the Linux-based system. Packets which come to the proposed switching node are handled primarily on the hardware, and only in case of need for complex operation execution are redirected to the software.

Stateful packet processing - was applied in [146] for solving the problem of network failure. The ability of OpenState [149] to respond to packet-level events has been utilized to determine the fast path recovery mechanisms for the relocation of flows affected by the network failure.

The connection monitoring workspace, called StateMon, is proposed and implemented in [144]. To keep the data plane as simple as possible, only the table of an open connection, based on OpenFlow match-action abstraction, is added to the end of the switch pipeline. The rest of the logic, which is in charge of maintaining the global status table and the state management tables, is implemented in the controller. Due to the OpenFlow communication limitations, the switch has been extended with a new protocol for the open connections table programming. Using StateMon, the stateful firewall and the port knocking application were implemented.

Because the current abstractions of the SDN's data plane, for the detection of network failures, did not provide the ability for fine tuning the detection mechanism in the switches, the new data plane design, called SPIDER, is proposed in [147]. SPIDER is a pipeline similar to OpenFlow which allows: (a) failure detection mechanisms based on periodic link checking on switches, (b) rapid redirection of traffic flows even in the case of remote failures, regardless of the availability of controllers. A model of flow structure based on the stateful data plane abstraction such as OpenState and P4, and the corresponding behavior model, are presented.

The feasibility of using the abstraction of a programmable data plane in the *iptables* offloading from the server processor to the smart network card is investigated in [148].

On a smart network card, based on the NetFPGA-SUME platform, an open packet processor based on the XFSM has been implemented.

F. SUPPORT FOR VARIOUS NETWORK TECHNOLOGIES

Since the inception of SDN idea, the research focused on Ethernet and IP network technologies. However, a significant need for the expansion of the SDN, to support other network technologies such as optical circuit switching (OCS), optical packet switching (OPS), gigabit passive optical networks (GPON), data over cable service (DOCSIS) and so on, has been noted. Some of the research focused on introducing reconfigurability into the data plane architecture, especially in the domain of the physical layer, while others opted for data plane abstraction to make it independent of lower-layer technology.

New structure of flow table - is proposed in [150] as an extension of OpenFlow v1.0 to support MPLS technology. The proposed extension allows an OpenFlow switch without the ability to route IP traffic to forward MPLS traffic. This is accomplished by having three packet header modification operations implemented in the switching node of the data plane:

1) push - adding new labels to the MPLS label stack,
2) pop - removing labels from the MPLS label stack,
3) swap - replacing the label at the top of MPLS label stack with a new label.

The prototype is implemented on the NetFPGA platform.

An extension of OpenFlow v1.1 architecture to support switch management in multi-technological transport layers is presented in [151]. The circuit flow table, which is not used for search, but contains information about existing connections, is proposed. This enabled support for hybrid switches that perform both the circuit and the packet switching. The proposed extension can be used for a smooth migration to fully packet-optical integrated nodes where, for example, packet routers would contain reconfigurable optical add-drop multiplexers (ROADM).

Reconfigurable architecture - called Open Transport Switch (OTS), which enables packet-optical cross-connections (XCON) and allocation of bandwidth on the optical element, is proposed in [152]. OTS consists of the following building elements: (1) a discovery agent which is in charge of detecting and registering resources, (2) a control agent which is in charge of surveillance and propagation of alarms and notification to the controller, and (3) the data plane agent which is responsible for programming the data path of the network element. Data plane entities can be time slots, XCONs, or MPLS labels. OTS is integrated into the SDN framework using the OpenFlow protocol.

The first demonstration of a fully-programmable space division multiplexing (SDM) optical network consisting of three architecture on demand (AoD) nodes interconnected by multi-core fibers (MCF) is presented in [153]. AoD nodes dynamically implement a node architecture based on the traffic requirements and consist of an optical backplane that interconnects MCF/single mode fiber (SMF) inputs, modules such as a spectral selective switch (SSS) or EDFA (erbium-doped fiber amplifiers) and MCF/SMF outputs. Later, a programmable FPGA-based optical switch and interface card (SIC) replacing traditional NIC and allowing direct interconnection of servers via optical top-of-the-rack (ToR) switches is proposed in [154]. Additionally, SIC enables the aggregation of the OCS and the OPS within the AoD.

The SDN-enabled OPS node for reconfigurable DCN is represented in [155]. The OpenFlow protocol has been extended to support wavelength management, management of spatial and time switching elements, and flow management of the OPS node. The data plane of the OPS node is modular allowing the constant reconfiguration time (in the order of nanoseconds) regardless of the number of node ports. Optical flows generated by the ToRs contain an optical tag by which an FPGA-based controller performs an OPS table lookup and determines the OPS port to which flows are forwarded. Given the nature of statistical multiplexing, contention between input signals from the same ToR is possible. Therefore, between the FPGA-based OPS and ToR controller, bidirectional flow control with ACK and NACK signals has been established. To enable communication between OPS and OpenFlow controllers, an OpenFlow agent has been implemented.

The SDN-based integration of time and wavelength division multiple access - passive optical metro networks (TWDM-PON) is demonstrated in [156]. The proposed solution is based on a simple OpenFlow controller that runs on one node where network nodes are implemented using additional cards. The prototype, made of two optical network units and two optical service units of the 10Gbps Ethernet-based TWDM-PON, is implemented using two Altera FPGA Stratix IV GT chips. It has been experimentally demonstrated that such an architecture can achieve a reconfiguration time of the node in a data plane below 4 ms.

The NEPHELE network architecture, featured in [157], has a scalable data plane built on verified commodity photonics technology. The NEPHELE data plane works in the time-division multiple access (TDMA) mode, where each slot is dynamically reserved for one communication on the rack to rack relation. Building blocks of the data plane are ToR and *pod* switches. ToR switches connect devices within the datacenter rack and in their northern ports have specially-configured optical transceivers working in TDMA mode. *Pod* switches, interconnected by the wavelength-division multiplexing (WDM) technology in the ring topology at the top of NEPHELE architecture, enable interconnection of all the ToR switches in the star topology. Within the *pod* switches, switching is performed using the array waveguide grating routers. The integration of the data plane of NEPHELE architecture into the OpenFlow-based SDN architecture is enabled through the implementation of the support for three types of interaction:

1) ability to advertise a device in the data plane (e.g., available wavelengths, active ports, available time slots),

2) operational configuration of the device (e.g., adding records to the flow table, creating cross-connections),
3) data plane monitoring, including asynchronous notifications and statistics download.

In a practical example, mentioned interactions are supported by implementing an SDN agent that acts as a proxy between the switches and the controller.

New data plane architecture - called SplitArchitecture [158], allows flexible mapping of control plane layers to data plane layers in a hierarchically structured model. Additionally, SplitArchitecture separates the forwarding and processing functionalities of the data plane element. The ability to process data in the data plane enables the implementation of OAM functionality as support for tunneling in carrier networks (PPPoE, pseudo-wire emulation, etc.).

A programmable and virtualizable all optic network infrastructure for DCN is presented in [159]. Its architecture is hybrid, and the data stream consists of SDN-enabled OPS and ToR switches connected to the SDN-enabled optical backplane with support for multicast circuit and packet switching. The optical backplane includes optical function blocks such as wavelength selective switches (WSS), OPSs and splitters, which can be dynamically configured to the arbitrary topology according to the requirements of a specific application. For intra-rack server interconnection, a programmable FPGA-based OPS/OCS hybrid NIC is used.

Since OpenFlow specifications did not include optical layer constraints at the time, the use of hybrid switches was discussed in [160]. A typical core network transmits a combination of packet and circuit services. It is implemented as one of the following multilayer architectures:

1) layered architecture in which the packet switching is positioned above the circuit switching,
2) parallel architecture in which circuit and packet switching are located at the same level of the hierarchy,
3) hybrid architecture in which a hybrid switch provides complete flexibility in the aggregation of circuit and packet switching on individual wavelengths.

This flexibility is enabled by the use of software-defined transceivers and the flexible network of ROADMs.

One approach to the integration of optical networking devices into an OpenFlow-based SDN architecture is presented in [161], [162]. The architecture of the software-defined optical network (SDON) with a hierarchical data plane structure is proposed. The data plane is divided into four layers to achieve more accurate resource management. Integration was achieved by introducing transport controllers between data and control plane. The task of the transport controller is an abstraction of the optical network infrastructure to the unified controller as a set of virtual resources, using the techniques of inter-layer provisioning and resource adaptation.

Hardware abstraction layer (HAL) - architecture, presented in [163]–[165], consists of two parts: a cross-hardware platform (CHPL) and a hardware-specific layer (HSL). CHPL enables virtualization and realization of OpenFlow mechanisms, independent of the hardware platform below, with an efficient pipeline. The pipeline is processing packet abstractions rather than actual packets, where packet abstraction contains a reference to the actual packet stored in the network device memory. HSL is a set of hardware drivers which implement primitive network instructions, specific to different hardware platforms. Examples of HAL architecture adaptations are presented for three groups of network devices: (1) optical devices, (2) point to multi-point devices and (3) programmable platforms. Therefore, an example of the dense wavelength division multiplexing (DWDM) ROADM optical switch for optical devices is presented. In this example, the optical fiber specificities are abstracted by a cross-connection flow table containing information about the input port, the output port, and the wavelength at which the established communication is performed. Additionally, an adaptation of the proposed HAL architecture for GPON technology is presented in [166]. As an example one on multiple architectures, adaptation to the DOCSIS system is provided in [164]. In this case, the entire data plane of the DOCSIS system consisting of the cable modem terminating system, cable modems and residential gateways is abstracted as a one aggregated switch. Since the DOCSIS platform is a closed system and configuration is only possible through standard interfaces, a DOCSIS proxy is introduced between the OpenFlow Controller and the DOCSIS system that performs described abstraction. Technical details of the DOCSIS proxy implementation are presented in [167]. Adaptation of HAL architecture for programmable platforms is provided for an EZappliance platform, based on the EZchip NP-3 network processor, and the NetFPGA board.

All-optical DCN architecture is presented in [168]. The data plane of the proposed architecture consists of OCSs based on large-port-count fiber switches, above which the MCF switch is settled. MCF switch forwards traffic between data centers. Two layers of OCS combined with SMF and MCF devices, under SDN control, form a flat DCN architecture. Besides MCF switches, additional components (such as FPGA based high-speed TDM switches and NICs) are placed in the data plane to meet different requirements. Above the presented data plane, OpenStack-based virtualization has been established, which simplifies the process of provisioning and SDN-based network management.

G. NETWORK AND NETWORK FUNCTIONS VIRTUALIZATION

The network functions virtualization (NFV) relies on host and network virtualization technologies, which enable the mapping of entire classes of network node functions to building blocks. Interconnecting (chaining) of building blocks creates complex communication services (e.g., firewalls, IDSs, traffic caches). Network functions virtualization complement the SDN so that it enables the organization of elements on the path through the data plane. Because of an unbreakable connection between NFV and SDN, research in the field of virtual networks and NFV has also implicated changes in

the data plane of the SDN. An overview of these research is provided below.

Reconfigurable architecture - for virtualized middlebox acceleration, called OpenANFV, is presented in [169]. From top to bottom, OpenANFV's architecture consists of: (1) an OpenStack orchestration platform, (2) a virtual network function (VNF) controller, (3) a NFV infrastructure, and (4) a network functions acceleration platform (NFAP). In short, when a specific middlebox needs to be realized, each VNF that builds it is instantiated as a VM node. The VNF controller takes care of the resource management required by each VM. NFAP is implemented using a PCIe card that contains a FPGA chip with static and partially reconfigurable (PR) regions. Functionalities of packet classification and switching are implemented in the static region. When required, accelerators are implemented using PR and connect to a switch in a static region.

The use of FPGAs as a platform for the NFV is proposed in [170]. In the proposed conceptual solution, FPGAs can be dynamically configured to provide hardware support to a specific application or another hardware. The platform should support hardware accelerators from different vendors, by defining a standard interface between these modules (NFV-IF). The NFV controller would perform the configuration of individual modules on the FPGA via the NFV configuration interface (NFV-Config-IF).

Data plane programming - using P4 was used to create a portable virtualization platform HyPer4 [171]. HyPer4 consists of a P4 program, named *persona*, running on a network device, a compiler and a data plane management unit. Persona is a generic P4 program that can be dynamically configured to mimic the functionality of other P4 programs through three phases:

1) parsing and setup,
2) match-action emulation,
3) egress phase.

In the parsing and setup phase, a packet is received and prepared for the processing according to the specifications of the P4 program, which is emulated in the second stage. The output phase manages output-specific primitives and prepares the packet for sending. To be able to emulate arbitrary P4 programs, the persona supports:

- programmable parsing,
- arbitrary definition of the packet field representation,
- matching to arbitrary fields,
- actions which can be complex collections of P4 primitives,
- virtual networking based on the recirculation of packets from one virtual device to another.

As proof of the proposed concept, using HyPer4 are emulated L2 Ethernet switch, IPv4 router, ARP proxy and firewall.

New data plane architecture - based on ForCES network elements with support for virtualization and programmability is proposed in [172]. A virtualization support in ForCES network elements has been achieved by introducing virtual machine technology in CE and FE. The virtual ForCES router, called vForTER, is designed as a proof of the proposed concept. The vForTER architecture consists of virtual control elements (vCE), virtual forwarding elements (vFE) and a particular FE called switching element (SE). SE is in charge of managing the virtualization of CE and FEs and the internal scheduling of traffic between virtualized elements. Also, an FE algorithm for the dynamic allocation of FE resources is executed on the CE. In the proposed vForTER design, the data plane consists of one FE and two vFEs. vFE performs the processing of packets received from the SE, using functions provided by LFBs. The vForTER prototype was implemented using VMWare ESXi hypervisor and the Click modular router.

A virtual filtering platform (VFP), based on a programmable virtual switch, is presented in [173]. The VFP is running on a Hyper-V extendable switch on the Microsoft Azure cloud platform and consists of match-action tables (MAT) and the packet processor. MATs are realized as layers that support configuration using a programming model and the operation with multiple controllers. The programming model is based on the configurable hierarchy of VFP objects: (1) ports on which the filtering policies are performed, (3) rules as MAT records, and (4) rule groups inside the single layer. By testing VFP on over a million hosts, over a period of 4 years, there have been several conclusions regarding data plane design:

- the design should be conceived as a stateful from the beginning,
- a precise semantic of the forwarding table layering is needed,
- physical layer protocols need to be separated from the data plane,
- all operations should be modeled as actions (e.g., tunneling as encapsulation and decapsulation),
- forwarding should be kept simple,
- commodity NIC hardware is not ideal for the SDN.

Virtual data planes, as described in [174], can be realized as software objects called virtual network objects (VNO), to which multiple network infrastructures can be mapped. The network slice can be created as needed by configuring the logical network via VNO, which liberates users from the need for expert knowledge in the field of virtual networks.

Hardware abstraction layer (HAL) - model for data plane resources orchestration based on UNIFY BiS-BiS [175] is presented in [176]. In the proposed model, each hardware element which affects delay and bandwidth must be taken into account, e.g., processor cores, memory modules, physical or virtual network interfaces, etc. Fast and efficient resource orchestrator (FERO) generates an abstract model based on hardware infrastructure during a bootstrap process. Incoming network service requests are mapped to available resources using the generated graph model. The prototype of the proposed solution was implemented using Docker and various software switches with added support for DPDK.

TABLE 2. Correlation of treated problems and problem-solving approaches.

Problem-solving approach \ Treated problem	Performance	Energy consumption	Quality of service	Measurement and monitoring	Security and reliability	Support for various network technologies	Network and network functions virtualization	Key limitations of ForCES and OpenFlow data plane architectures	Generalization of problem-solving approaches
Data plane programming				•			•	There is no ability to use the language to describe and program the data plane	A
Stateful packet processing	•		•	•	•			There is no ability for the state-aware or context-aware packet processing	B
Reconfigurable architectures	•					•	•	There is no ability for reconfiguration of functionality or data plane parameters	C
Physical layer management		•						There is no ability for process management below the flow table lookup level	C
New structures of flow tables	•	•	•			•		There is no ability to change the table flow structure (for example, when adding support for new protocols)	C/D
New mechanisms for flow tables lookup	•	•						There is no ability to implement new algorithms for flow table lookup, or enhance existing ones	C/D
New packet classification mechanisms	•				•			There is no ability for higher layer protocols' packets classification (e.g., TCP, HTTP, SIP)	B/C/D
Hybrid architectures (SW-HW or HW-HW)	•		•	•	•			There is no ability to offload data plane processes from hardware to software or vice versa, and to accelerate processes with additional hardware	D
New data plane architectures			•	•	•	•	•	There are general limitations on the topology flexibility and data plane functionality	D
Hardware abstraction layer		•	•			•	•	There is no direct support for specific network technologies (e.g., DOCSIS, GPON, OCS/OPS, PPPoE) — data plane implementations depend on the physical layer network technology or a target platform	D

KEY FACTORS FOR DATA PLANE EVOLUTION

By reviewing the research which had addressed different problems by the categories listed in this section, we observed several common problem-solving approaches. Then we established a correlation between treated problems and problem-solving approaches. Afterwards, we identified the key limitations of ForCES and OpenFlow data plane architectures which have conditioned the specific problem-solving approach. Based on identified key limitations, we generalize the approaches to addressing the problem of programmability and flexibility of the SDN's data plane in four categories:

A) data plane languages,
B) stateful data plane,
C) deeply programmable networks,
D) new data plane architectures and abstractions.

Table 2 shows the correlation of treated problems and approaches to addressing an issue of programmability and flexibility of SDN's data plane, as well as a generalization of problem-solving approaches based on identified key limitations.

From the motivations and results of the research presented in this section, we can conclude that the perceived limitations of ForCES- and OpenFlow-based data plane architectures cannot be relatively simply overcome. The programmability of the internal data plane structure is imperative for the future flexibility of the SDN from the aspects of flows, functions, resources and topology.

VI. GENERIC APPROACHES TO IMPROVE THE DATA PLANE FLEXIBILITY AND PROGRAMMABILITY

Before entering a critical review of the research which through the four generic approaches achieved the improvement of the programmability and flexibility of the SDN's data plane, we will make a brief recapitulation of the work done so far. In the first sections, an overview of the ForCES and

OpenFlow data plane architectures, as well as its hardware- and software-based implementations, is provided. Afterwards, an overview of SDN-related research is presented, whose results implied the evolution of data plane, under which we meant a gradual deviation from the original architecture given by ForCES and OpenFlow specifications. By establishing a correlation between problems treated by SDN-related research and problem-solving approaches, we identified key limitations of ForCES and OpenFlow data plane architectures. Based on identified key constraints, we generalized approaches to addressing the problem of programmability and flexibility of the data plane, into four categories which represent the organizational units of this section.

A. DATA PLANE LANGUAGES

By contemplating research which dealt with the definition of data plane language, we have recognized two categories:

1) data plane description languages,
2) data plane programming languages.

1) DATA PLANE DESCRIPTION LANGUAGES

Data plane description languages enable the description of the data plane structure and its components (e.g., an order of elements in the packet processing pipeline, parameters of elements). A review of the most relevant research from this category is given below.

The high-level language for programming packet processors, called P4, is proposed in [177]. When designing the P4 language, three goals have been set: (1) to enable on-demand reconfigurability of parsing and processing the packet, (2) to ensure protocol independence by enabling the specification of the packet header parsing and the match-action table, and (3) to achieve independence from the target platform. For the definition of P4 language, an abstract forwarding model was used in which the switch forwards packets to the pipeline through the programmable parser. The pipeline is composed of multiple match-action module stages, which can be arranged in a parallel, in a series or in a combination. The P4 program contains the following components:

- packet header definition,
- parser definition,
- match-action tables,
- constructions of actions made of simple protocol-independent primitives,
- control programs which determine the order of application of the match-action tables on the packets.

In short, by using the P4 language the programmer defines the packet processing mode in the data plane without taking into account the implementation details. The written P4 program is compiled into a table dependency graph which can be mapped to a specific software or hardware switch. The design of the P4 compiler for reconfigurable matching tables (RMT) [178] and Intel FlexPipe programmable ASICs is presented in [179]. A hardware abstraction, which is common for both chips, is defined to achieve the independence of the compiler from the target platform. The physical pipeline of the chip is modeled as a directed acyclical graph of processing stages, while the memory is abstracted with match tables which support records corresponding to the type of memory (e.g., TCAM as a ternary match table, SRAM as an exact match table).

Considering existing P4 language constraints when describing the DCN data plane, the P4 language extension is proposed in [180]. Additional constructions were introduced: (1) cloning the packet, (2) rejecting the packet, (3) generating Digest, and (4) adding CRC-16 hash to a specific packet field. It has also been shown that existing P4 language constructions enable the description of a significant number of DCN switch functionalities.

The PISCES, a P4 programmable software switch presented in [181], is a modified version of OVS in which parsing, matching, and action execution codes are replaced with a C code generated by a P4 compiler. In order to customize the OVS function to the P4 principles, additional changes were made in the OPS: (1) support for arbitrary encapsulation and decapsulation was provided by adding two new primitives to manipulate the header of the packet, (2) support for conditioned action execution has been added, and (3) support to the generic verification and checksum update mechanism was provided. The PISCES compiler generates the source code of the OVS software switch based on the P4 program, which then needs to be compiled into executable files of the switch.

Since FPGA technology has become popular in prototype network hardware, the P4FPGA framework which enables compilation of the P4 program into the FPGA firmware is proposed in [182], [183]. P4FPGA generates the appropriate BSV code suitable both for simulation and synthesis, based on the P4 program. Generated BSV code contains a description of the P4 pipeline and additional supporting infrastructure. The supporting infrastructure includes FPGA memory management and the pipeline communication with other peripheral units on the target platform (e.g., PCIe interconnection, 10G Ethernet interface).

In [184], Xilinx, one of the leading manufacturers of FPGA chips, has presented its contribution toward the implementation of reconfigurable network elements in both control and data plane in the form of the development environment SDNet. It allows the specification of packet switching node elements in a domain-specific language (e.g., P4), translation into hardware description languages (HDL), like VHDL and Verilog, and its synthesis and deployment on a broad range of FPGA and SoC chips. The basic features of SDNet environment include:

- generating custom hardware components to perform specific tasks (e.g., editing, searching, parsing),
- generating a specially customized data plane hardware subsystem according to the application or the user requirements,
- generating a particularly customized firmware for designed SDNet hardware architectures,
- generating testbench for validation and debugging.

As an example of using the SDNet environment, the next generation NIC architecture, called the Softly Defined Line Card, has been presented. In the card architecture, the customized data plane consists of a packet processor and a programmable traffic controller. Also, IP cores for MAC, forward error correction (FEC), physical coding sublayer (PCS) and switching functionality can be used from the extensive network SmartCORE library. Other standard interfaces (e.g., external memory interfaces) are available from the LogiCORE IP library.

An intermediate data plane language, called NetASM, is proposed in [185]. NetASM is conceived as a link between high-level languages (e.g., P4) and a diverse and growing set of hardware platforms. The instructions of NetASM languages are executed sequentially or concurrently on packet-related states, or on persistent states at the pipeline level. By introducing a packet-related state, it is possible to parallelize the pipeline. The NetASM language instruction set contains 23 instructions and enables: (1) data loading, (2) data storing, (3) calculation, (4) branching, (5) operations on the packet header, and (6) special operations such as hash and checksum.

2) DATA PLANE PROGRAMMING LANGUAGES

On the other hand, data plane programming languages enable the implementation of new algorithms in the data plane (e.g., new packet scheduling mechanisms, new packet classification methods), and the most important representatives are Protocol-oblivious Forwarding (POF) [186], and Domino [187].

POF does not need to know the packet format. Its only task is to extract the search keys from the header of the packet according to the controller's instructions, search the table, and then execute associated instructions. The instructions are given in the form of an executable compiled from the code written in the Flow Instruction Set (FIS) language. FIS instructions allow manipulation of packet headers, forwarding tables contents, and statistical counters. Thus, support for new protocols is provided without changes in the data plane.

Domino is an imperative language with a syntax similar to C language and allows writing programs for packet processing using packet transactions. Packet transactions are inseparable sequential code blocks. The execution of Domino packet transactions is foreseen on the new machine model of the programmable line-rate switches, called Banzai. The Banzai machine architecture consists of an ingress and an egress pipeline. Parsing a packet is out of the scope of the Banzai model, and it is assumed those packet headers which are entering pipeline are already parsed. The pipeline consists of many stages that contain atoms, i.e., vectors of programmable units for packet header processing. The Domino compiler extracts the code fragments, which are executing atomically, from the description of the algorithm and maps them to the appropriate configuration of atoms in the Banzai machine.

REMARK ON FLEXIBILITY AND PROGRAMMABILITY

In the first category - data plane description languages - P4 is particularly emphasized, which treated limited SDN's data plane flexibility by increasing the programmability of parsing, matching and action processes. Protocol independence was achieved by introducing reconfigurability in the packet parsing and processing, and target platform independence was achieved by hardware abstraction. However, in this way, some packet processing functionalities, such as queuing, scheduling and physical layer management, are neglected, limiting flexibility in term of forwarding function operation. As P4 does not envisage the use of hybrid architecture as a target platform, its flexibility in terms of function scaling and function placement is significantly limited.

On the other hand, the alternative to the P4 language are hardware description languages, which provide high programmability but with a complicated and time-consuming development process. Although such a limitation can be partially overcome using compilers which translate domain-specific languages into HDL, domain-specific languages should retain sufficient details specific to target hardware which would allow the programmability of all data plane processes.

The second category includes data plane programming languages in term of implementation of arbitrary packet processing algorithms directly in the data plane. Such languages achieve high flexibility in term of function operation through the programmability of all data plane processes. However, their dependency on the destination platform (specific packet processors or packet machines) limits the flexibility from the aspects of function scaling and placement.

Therefore, as open problems which should be addressed by future research, we highlight the following:

1) the lack of adequate constructs of current languages which would allow complete programmability of all data plane processes, and
2) the inability of the existing languages to adequately abstract the details specific to the destination platform, which would ensure independence from the destination platform without losing the granularity in programming the data plane processes.

B. STATEFUL DATA PLANE

As already noted in the Section V, an approach based on the introduction of finite state machines into the data plane has been applied to solve a significant number of data plane problems. A review of the most important research aimed at generalizing the introduction of state-aware packet processing into the SDN's data plane is provided below.

A stateful forwarding abstraction (SFA) in SDN's data plane is presented in [188]. The co-processor unit, called Forwarding Processor (FP), is implemented within the SDN switch using the CPU. Extended OpenFlow instructions are used to redirect packets or flows from flow tables to the FP, in which complex processing functionalities are

implemented. This has enabled stateful network processing on the higher layers of the protocol stack.

The new stateful datacentre architecture (SDPA) is proposed in [189] as a follow-up of the research presented in [188]. Unlike the standard match-action OpenFlow paradigm, a new ''match-state-action'' paradigm has been proposed in which state information is maintained within the data plane without the heavy involvement of the SDN controller. The SDN switch architecture, which supports SDPA in such a way that the FP, the state table and the policy module are added to the standard architecture of the SDN switch, is proposed. FP's task is to maintain the flow or the packet states. The policy module is used to customize and manage processing policies issued by the controller. Based on the proposed architecture, hardware and software prototypes of SDPA switches and application examples such as stateful firewall, defense against the domain name system (DNS) reflection attack and network address translation (NAT) functionality have been implemented.

Although the OpenFlow architecture is limited regarding programmability within the switch, a non-trivial subclass of stateful control functions, which can be abstracted as Mealy's FSMs, and are already compatible with OpenFlow hardware from version 1.1 with minimal architectural changes, is advocated in [190]. The proposed approach, OpenState [149], focuses on introducing programmable states and transition to the OpenFlow. The control logic uses packet-level events as triggers for the change of forwarding rules at wire speed inside the device itself. OpenState introduces the stateful block as an extension to a single flow table and can implement: (1) state table associated with flow identities, and (2) the XFSM table which performs a search based on the state label and packet header fields, and returns the associated forwarding action and the next state label. Stateful blocks can be chained into the pipeline with other stateful blocks as well as the classic OpenFlow tables. The proposed abstraction generalizes the OpenFlow match-action rules by using XFSMs which are directly executed within the switches, thus offloading controllers and creating abilities for complex control operations at the packet level in a fast data plane.

The contribution to the improvement of the data plane programmability by introducing stateful packet processing within network switches is presented in [191]. The proposed solution is called an Open Packet Processor (OPP), and its primary goal is to provide an ability for direct packet processing in the fast path, with efficient storage of flow state informations. OPP is based on the abstraction of the match-action phase of the OpenFlow using XFSM. The forwarding evolution is described by the FSM, where each state defines the forwarding policy, and the packet-level event initiates the transition to the next state. The workflow of the OPP architecture consists of four phases: (1) flow context table lookup, (2) conditions evaluation, (3) XFSMs execution, and (4) update of registers and flow context tables. The OPP prototype was implemented using the NetFPGA-SUME platform, where SRAM and TCAM were utilized for the flow context table and XFSM implementation, respectively. Later in [192], the same authors have presented several cases of OPP usage in the implementation of advanced network functions:

- packet forwarding based on load balancing,
- topology discovery based on L2 data,
- load balancing in the private network with static NAT.

REMARK ON FLEXIBILITY AND PROGRAMMABILITY

The introduction of the match-state-action paradigm in the data plane has, through the increase in programmability of actions, improved the flexibility of the data plane from the aspect of the function operation. In the proposed solutions, the control logic is dependent on the state of the data plane and driven by packet-level events. Since the control plane does not have the ability to monitor and manage the state of the data plane, the flexibility from the aspect of the function placement is limited. Additionally, the lack of adequate synchronization of the data plane state reduces flexibility from the function scaling aspect.

Although reviewed research has addressed the problem of introducing state-aware packet processing into the SDN's data plane adequately, we notice some key issues that have not been solved and which lead us back from the basic SDN idea - managing the data plane from the control plane. Therefore, we highlight the following open issues:

1) state monitoring and management,
2) data plane state synchronization.

C. DEEPLY PROGRAMMABLE NETWORKS

The deeply programmable network (DPN) is mentioned for the first time in [193], and implies a step further to increase the data plane programmability in the SDN. The DPN's goal is to enable advanced packet processing functions such as caching, transcoding, and DPI, and to support new protocols, through increased data plane programmability below the flow table configuration level.

1) ORIGINAL APPROACH TO THE REALIZATION OF DPN IDEA

Realization of the DPN idea has started through a VNode design which consists of slow and fast paths. The slow path represents a programming environment which consists of Intel Architecture servers, while the fast route performs network traffic processing using network processors. VNode relies on the generic router encapsulation (GRE) for tunneling of frames which come from different protocols. The extension of the VNode architecture with a physical node called Network ACommodation Equipment (NACE) is presented in [194]. NACE performs a dual role: (1) a gateway between a virtual network and an external Ethernet/virtual local area network (VLAN) network, and (2) a virtual switch between slices of the virtual network. To implement the data plane, NACE relies on 10Gbps Ethernet hardware including an L3 switch which supports VLAN and IP routing functions, and a service module card which performs conversion of

packet formats between external and slice-internal formats. The internal format is based on GRE, according to VNode architecture, while the external format is based on the VLAN. Due to VNode's limitations regarding the complexity of fast path programming and the inability to handle other frames besides the conventional Ethernet frames, a new DPN node design inspired by the POF idea, called FLARE node, is proposed in [195]. FLARE can handle arbitrary protocol frames thanks to the PHY designed to support any frame. This means that there is no need for traffic tunneling which reduces overhead and increases network bandwidth and performance. The FLARE node architecture has supported the implementation of the multiple switching logic within a single node by dividing physical node resources into many fully programmable slivers, through the virtualization technology. Interfaces to physical ports of the node, a sliver management system, and a classification machine named packet slicer are implemented at the lowest level of architecture. The sliver management system enables dynamic installation and removal of slivers. The packet slicer performs fast packet scanning and its multiplexing or demultiplexing from or to slivers. A programmable control and data planes and virtual ports are available to the user within one sliver of the FLARE node. The data plane consists of a fast and slow path. The fast path of the FLARE node is implemented using a Click modular router, which runs as a multithreaded application on multicore processors. The OpenFlow protocol support modules are implemented within a slow path of the FLARE node. Each sliver allows the implementation of arbitrary switch logic, e.g., one sliver can implement OpenFlow 1.0 and the other OpenFlow 1.3. The benefits of this approach are reflected in the abilities of instant replacement of the switch software, and the gradual upgrade of the network while maintaining compatibility with the legacy technologies. The general advantages of FLARE architecture are supporting the extension of SDN capabilities and the development of new protocols (e.g., non-IP protocols) in the research community. Later, improvements in VNode infrastructure using the FLARE node have been presented in [196]–[199]. Support for edge network virtualization and better resource management has been added. Different use cases of DPN-based NFV were also presented:

- application-specific traffic control,
- smart M2M gateways,
- custom actions for OpenFlow switches,
- content/information-oriented networks.

The use of the DPN in the application-specific slicing of the data plane of the mobile virtual network operators (MVNO) is shown in [200]. FLARE node is used for classification of traffic specific to the particular application or the device. In this way, it enables: (1) fine-grained QoS application, (2) traffic engineering based on the type of application, device, and other status information, (3) implementation of application-specific value added services, (4) support for intra-network security and parental control, and (5) improvement the bandwidth utilization based on statistical data on particular applications and devices. A context-aware IoT architecture based on the MVNO switch, as shown in [201], has been built on the ground of this idea. In the proposed architecture, data collected through different sensors are transmitted over the IoT gateways to MVNO switches, which then forwards the received data to the central service controller for further processing. MVNO switches are realized as a slice of the FLARE node, and for packet forwarding are using information from the application layer content of the packet. As a fresh example of using the FLARE node, it is also worth mentioning the concept of network slicing in the 5G mobile network [202]. FLARE is used for the implementation of eNodeB and evolved packet core (EPC) nodes. eNodeB is running as a virtual machine instance within a FLARE slice. EPC is also implemented as a FLARE slice, where forwarding and processing of user data (e.g., Serving Gateway and Packet Data Network Gateway) are implemented in the data plane and signaling entities (e.g., Mobility Management Entity) in the control plane.

The model of the L7 switch which forwards packets based on the application layer content using the regular expression, and offers a south-bound API for configuring regular expressions as needed during switch operation, is proposed in [203]. The switch is realized by implementing two additional Click elements on the FLARE node: (1) L7Classifier and (2) L7Register. L7Classifier performs the identification of packet flows based on the IP address and TCP port number, and packet forwarding to the output interface or the L7Register element based on the content of the flow table. L7Register performs the matching of the packet contents of the particular flow with a regular expression. When there is matching with the regular expression in the L7Register element, an output interface is being updated in a flow table. By cascading multiple L7Register elements, it is possible to implement more complex processing of the application layer content.

The TagFlow architecture for packet forwarding based on flow tags, which is implemented using a FLARE node, is proposed in [204]. TagFlow's architecture is based on a DCN which has two edges: the ingress edge and the egress edge. The ingress edge includes switches which connect the DCN's core with application servers and the egress edge is the point where packets leave DCN. The TagFlow working principle is as follows: (1) when the packet comes to the ingress edge, the edge switch performs the classification and tagging of the packet and forwards it to the next hop, (2) the packet is forwarded based on the tag to the egress edge, (2) the egress edge switch removes the tag from the packet and forwards it to the destination. The classification of the packet can be any application layer classification, where each traffic class can have its tag and be treated as a separate flow. The tag is added to the end of the packet (i.e., trailer tagging) to keep compatibility with current network technologies that are unaware that the tag exists. The tag-based packet forwarding is reduced to the forwarding table lookup on each core switch. The forwarding table contains ordered pairs <flow

identifier, action> as records, with supported actions similar to those from the OpenFlow.

The extension of TagFlow architectures with user-defined actions, designed to provide full flexibility and programmability of the SDN's data plane, is presented in [205]. The network operator describes user-defined actions using high-level languages on commodity hardware. Implementations of user-defined actions on TagFlow and OpenFlow architectures were presented, and experimental comparison of its performance was performed. The TagFlow-based solution was 33% faster than OpenFlow-based.

The application of the DPN concept to improving the performance of DPI-based routing of applications transmitted via UDP traffic is presented in [206], [207]. The proposed solution is demonstrated on two UDP-based applications: (1) file reading and (2) database access. Unlike the standard methods which perform destination servers' load balancing using the round-robin method, the proposed solution conducts the forwarding of application requests to servers based on the content of the request. In the first case it is the file name, and in the second case, the database name. Later, an upgraded DPN switching node solution based on the FLARE node, for TCP-based applications, was presented in [208]. The demonstration was done on the HTTP application, where the DPN switching node mediated in communication between the client and the server. By analyzing the content of a client's request, when possible, the DPN node generates HTTP responses to the client without server engagement, thereby improving the performance of the application.

2) OTHER APPROACHES TO THE REALIZATION OF DPN IDEA

Although the idea of DPN was developed by a group of researchers from the University of Tokyo, other researchers have also generated a significant contribution to this area. Thus SwitchBlade, a platform for the implementation of custom protocols on programmable hardware, is introduced in [209]. It enables the implementation of individual hardware modules on the fly without the need for hardware resynthesis. The SwitchBlade pipeline includes customizable hardware modules which implement the most common data plane functionalities. By combining these modules, it is possible to implement support for new protocols. For more complex tasks which can not be implemented on hardware, there is support for software processing based on packet rules or flow rules. Given the need to simultaneously run multiple protocols on the same hardware, the resource isolation mechanism was implemented by using separate forwarding tables. The prototype of the proposed solution is implemented on NetFPGA platform.

Another example of DPN idea realization is a split SDN data plane (SSDP), presented in [210] as a new switch architecture which combines non-flexible commodity switching chips and a deep programmable co-processor system to resolve the limitations of the SDN innovation potential. The SSDP architecture consists of two data paths: (1) a commodity switching chip which uses TCAM to forward aggregated flows, so-called macro-flows, and (2) NP units which perform micro-flows processing most often on L4 and higher layers. The SSDP prototype is implemented on the Dell PowerConnect 7024 platform which uses 24x1Gbps switching chips connected to the programmable subsystem (PS) via the XAUI interface. The PS is based on a 4-core microprocessor without interlocked pipeline stages (MIPS) running an OPS-based OpenFlow v1.0 software switch. The principle is that packets which do not match records in the TCAM of the switching chip are forwarded to the PS, where software flow tables are looked up. The OpenFlow controller determines where the particular flow will be written: in the TCAM on the hardware or in the software flow table.

The solution presented in [211] is based on the idea of an edge network node supporting user-driven data plane applications which can monitor and, if necessary, modify network traffic in transit. User-driven applications are intended to be executed within a network slice associated with the corresponding actor (e.g., network service provider, end user, content provider, etc.). The proposed edge network node architecture consists of the following components:

- software switch,
- network hypervisor,
- controller,
- embedded web server,
- network gateway,
- management server.

The software switch, implemented by OVS, performs packet forwarding based on the flow rules list. The FlowVisor-based network hypervisor performs virtualization and splitting of the network into so-called network slices, which allows connection of the node to multiple OpenFlow controllers. The controllers execute data plane applications on the encapsulated data obtained by the network hypervisor. Other components implement the functionalities of network node management and its connection to the rest of the network.

The implementation of a customizable and programmable PC-based switch, called the NetOpen switch, which supports different traffic processing functionalities within a data plane, is presented in [212]. The NetOpen switch architecture consists of interconnected processing modules and forwarding elements. The processing modules are working independently of the other modules, and by combining different modules, it is possible to create a data plane specially adapted for each traffic flow. Each processing module consists of a serially connected submodules: (1) pre-processing, (2) processing, and (3) post-processing. The forwarding element is responsible for the transfer of the packet between the network interfaces and the processing modules, following the service functionalities of the particular traffic flow. The topology and configuration of individual processing modules within the data plane are generated from the flow table. Hence, the flow table is used as a programmable interface between the data plane and the controller.

The Scalable Programmable Packet Processing Platform (SP4) based on the software router is proposed in [213]. The SP4 data plane architecture is a component-based pipeline supporting three types of components: (1) serial, in which packets are processed in strict FIFO order, (2) parallel, in which concurrent threads can process multiple packets, and (3) parallel, context-oriented components where packets are grouped by the context-key and within the group are processed according to the order of arrival in the system. The operation of the proposed solution is demonstrated in several use cases: analysis of simple mail transfer protocol (SMTP) traffic, interception of voice over IP (VoIP) calls, and DDoS attack detection.

A new pipeline architecture of the switching chip based on the RMT is proposed in [178]. RMT enables changes in the data plane without hardware modification. This is accomplished with the help of a minimal set of action primitives which specify how to handle the packet header in hardware through: (1) definition of arbitrary headers, (2) definition of arbitrary matching tables over arbitrary header fields, (3) definition of the packet header modification mode, and (4) maintenance of the states associated with packets. The architecture consists of a large number of stages of the physical pipeline, mapped by the logic stages of the RMT, and according to the resource needs of each logic stage. The components of the physical pipeline are the configurable parser, the configurable matching memory, and the configurable action machine. These components enable the implementation of arbitrary match-action processing of the packet.

Inspired by the ideas of chemical reaction engineering, the realization of data plane functions using chemical algorithms (CA) was proposed in [214], [215]. It has been shown that CA can be easily and quickly modified and reprogrammed on FPGA hardware without the need for translation into an intermediate program or HDL code. By using CA, it is possible to give an expressive and straightforward representation of rule-based algorithms for network dynamics management, which is suitable for extending the functionalities of the SDN's data plane.

REMARK ON FLEXIBILITY AND PROGRAMMABILITY

We believe that applying the DPN paradigm can solve a wide range of problems caused by insufficient programmability and flexibility of the SDN's data plane. For example, a deeply programmable node FLARE, using virtualization techniques, positively affects the flexibility from the aspect of function scaling. By introducing deep programmability into the Click-based software path, FLARE node achieves high flexibility in term of forwarding function operation.

However, one group of researchers focused on the use of conventional servers eventually equipped with special network processors, while others decided to use FPGA technology. That partially limits the flexibility in term of the forwarding function placement. Consequently, we claim that all reviewed DPN architectures have one common disadvantage - they are not independent of the target platform, which remains an open problem for future research.

D. NEW DATA PLANE ARCHITECTURES AND ABSTRACTIONS

In the last category of generic approaches to improve the programmability and flexibility of the SDN's data plane, an overview of research which generated new architectures or abstractions of existing data plane architecture is given. Although approaches to creating new data plane level architectures are diverse, they share the same motivation - overcoming limitations imposed by the original architecture of the OpenFlow-based SDN.

1) NEW DATA PLANE ARCHITECTURES

The conceptual solution presented in [216] is based on introducing a new component called *network fabric*. Network fabric has been defined as a group of forwarding elements whose primary function is the transmission of the packet. The working principle of the proposed solution can be described in the following steps: (1) the source node sends packets to the input edge switch which, after providing network services, forwards packets to the network fabric, (2) the network fabric performs fast forwarding of packets to the egress edge switch, (3) the egress edge switch sends packets to the destination node. The network fabric is transparent to the end nodes. The management of edge switches and the network fabric is supported by the control plane.

Several dynamic scenarios which illustrate the main challenges associated with the development of a framework for software-defined middleboxes are presented in [217]. The representation of the middlebox state, the middlebox manipulation and the implementation of the control logic are listed as the main challenges. The abstraction which exploits the inherent mapping of the status to the value of the protocol header has been proposed for the representation of the middlebox state. The introduction of three basic operations has supported the manipulation of the middlebox state: (1) state retrieval, (2) state addition, and (3) state deletion. The control logic remains under the control of the SDN controller.

In [218] it is argued, through the use of practical application scenarios, that flow tracking is required to ensure the consistent implementation of network management policies in the presence of traffic dynamics. To achieve this, the extension of the SDN architecture has been proposed. In the proposed extension, middleboxes add tags to outgoing packets to provide the required context, and these tags are used by switches and other middleboxes for systematic implementation of the management policies. The context carried by the tag refers to middlebox-specific internal information which is critical for the implementation of management policies, such as the ratio of the number of hits and misses on the proxy or public-private NAT mapping. For the implementation of the proposed solution, switches must support the operations of inserting the tag into the header or the packet content and matching of the packet with the corresponding

tag, while the purpose of the tag remains in charge of the controller.

The SDN architecture which enables application-aware packet processing is presented in [219]. The basic idea is to intercept the packet before sending it to the controller and applying the application processing logic within the switch. The proposed solution has been implemented by extending the OVS with the support for special flow tables, so-called application tables, and application modules processing the traffic for which there is a matching within the application table. The pipeline of the OpenFlow data plane has been preserved with minor changes. Immediately after OpenFlow flow table lookup, packets for which there is no record in the flow table are forwarded to the additional application table lookup, instead of encapsulation and sending to the controller. Application modules can implement different functionalities such as firewall and context-aware load balancing.

2) DATA PLANE ABSTRACTIONS

On the other hand, the data plane abstraction motivation is established on the fact that a majority of data plane implementations depend on the physical layer network technology or the target hardware- or software-based platform. Some approaches to the data plane abstraction are presented below.

New hardware abstraction, called Programmable Abstraction of Datapath (PAD), for various network devices such as access devices, network processors, and optical devices, is proposed in [220]. PAD enabled data plane behavior programming through protocol and function header definitions, and by using generic byte-oriented operations. The PAD architecture consists of several functional elements:

- ports,
- search engines,
- search structures,
- execution engines,
- forwarding functions.

The PAD's working principle is as follows: (1) the received packet is merged with metadata and forwarded to the search engine, (2) after a successful search, the packet metadata together with the search results are forwarded to the execution engine, (3) the execution engine calls the function on the packet based on the results of the preceding search, and finally (4) the packet is forwarded to the output port. Multiple packet processing through the above steps can be implemented using loopback ports. PAD configuration management is possible through the PAD API on the northbound interface. The PAD API functionalities are as follows: (1) retrieval of information about data plane capabilities supported by specific hardware, (2) managing search structures, functions, and network protocol definitions, and (3) adding and deleting records in search structures. Specially designed languages such as NetPDL and P4 can be used to define network protocols and operations running on the execution engine.

The Network Abstraction Model (NAM) is proposed in [221] as a unique model for packet forwarding and network functionalities in SDN and NFV. The proposed model should enable the control, management, and orchestration of processes and network functionalities in the data plane, with the help of a unique protocol. It has been noted that the network device has a forwarding plane based on building blocks (BB) which together create required functions and the operating plane which is in charge of maintaining the state of the device. Based on the observed, ForCES was selected as the framework for the realization of the abstraction model. Additionally, the ForCES LFB-based model does not see the difference between a physical and a virtual device, which makes it convenient for NFV abstraction. The architecture of NAM is organized in such a way that BBs map to LFBs, and VNFs to one or more network devices.

TableVisor, a transparent proxy layer which enables pipelined packet processing and extension of hardware flow tables using multiple hardware switches, is proposed in [222]. Pipelined packet processing is enabled by emulating single multi-table switch using multiple hardware switches. On the other hand, emulation of large hardware tables by combining TCAM memory from multiple switches is also supported.

REMARK ON FLEXIBILITY AND PROGRAMMABILITY

New architectures, analyzed in this subsection, have tended in various ways to improve the flexibility of the data plane. In some research, a separation of the edge functions from the core forwarding functions influenced the flexibility from the aspect of function scaling. In other studies, by introducing a context into the packet processing process and using middleboxes, they improved flexibility in term of function operation.

Also, different data plane abstractions have contributed to flexibility. Thus, hardware abstractions enabled greater flexibility in term of function scaling, while network functionalities abstractions had a positive effect on the flexibility from the aspect of the function operation.

However, neither approach has given satisfactory improvements to all of the considered aspects of flexibility, which remains an open problem which should be addressed by future research.

VII. FUTURE RESEARCH DIRECTIONS AND CONCLUSION

Considering that previous surveys of the SDN-related research did not focus sufficiently on the data plane, this paper provides a comprehensive survey of the wired data plane in the SDN. The prerequisites for advancing the development of SDN's data plane are created through the proposal of future research directions which adequately address problems of programmability and flexibility. The complete process from identifying problems to the definition of future research directions, as shown in Figure 5, is carried out through the following steps:

1) An overview of actual SDN's data plane architectures is provided.

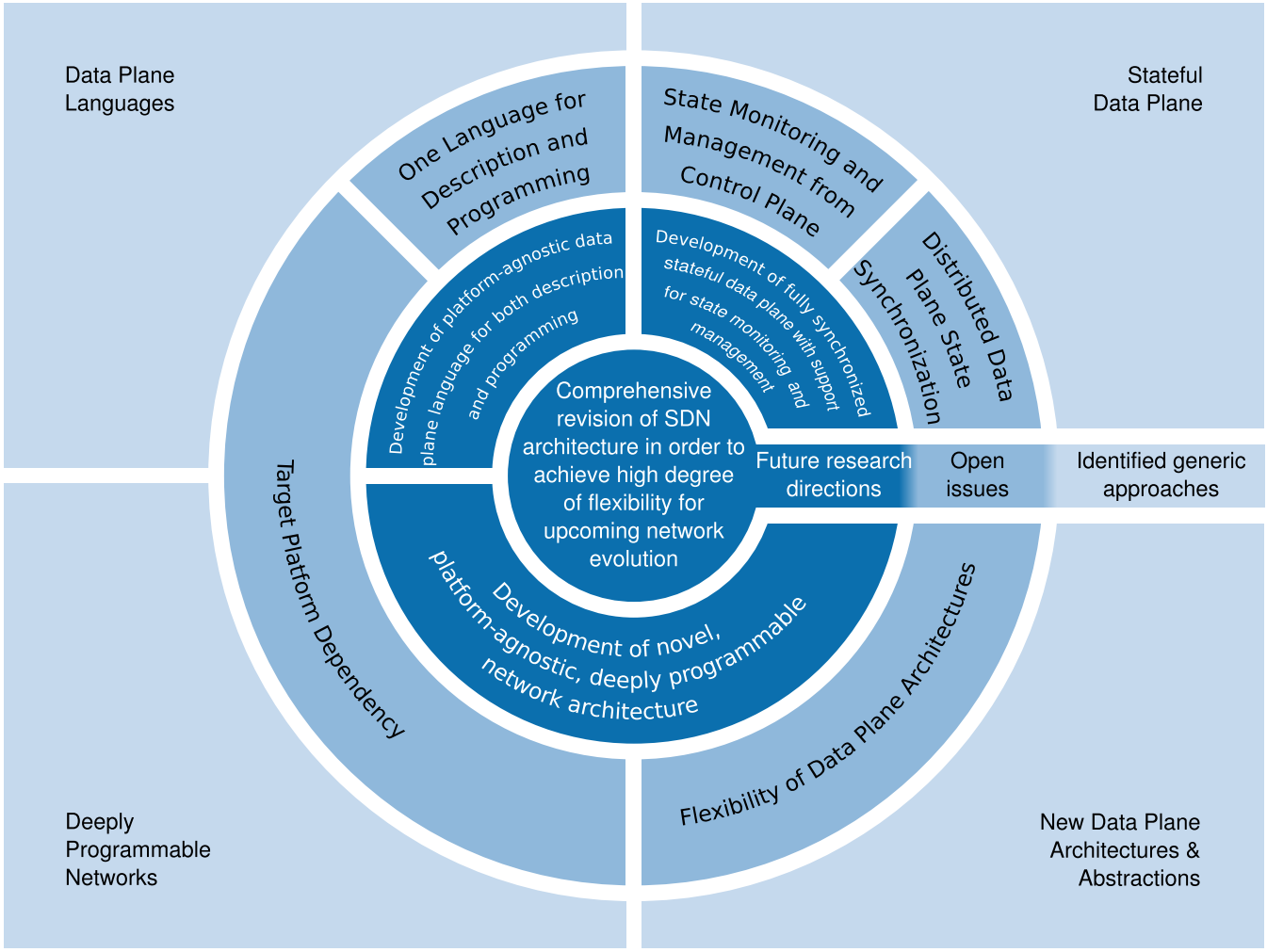


FIGURE 5. The pursuit of novel data plane architecture through future research directions.

2) An overview of software- and hardware-based supporting technologies which enabled data plane implementations is provided.
3) A review of SDN-related research with the aim of identifying key factors influencing the data plane evolution is given.
4) A critical review of generic approaches to improving the data plane flexibility and programmability is given.

In the realization of the first step, standardization of the SDN concept and architecture has been elaborated through the historical context. A particular attention is dedicated to the specification of ForCES architecture and to first attempts to realize SDN following that architecture such as NEon and Ethane. Since OpenFlow, albeit different from ForCES, emerged as a realization of the SDN idea, the most critical aspects of its architecture were compared with ForCES.

Afterwards, an overview of the definitions of network flexibility and programmability and some general considerations of flexibility in other domains is given. Then, a review of the constraints of ForCES and OpenFlow-based data plane architectures, through the considered definitions and aspects of flexibility and programmability, is presented. Given that a lot of reviewed research is established on the experimental evaluation, an overview of hardware- and software-based technologies which served as good support for data plane implementation is given.

To address problems of flexibility and programmability of the SDN's data plane, a lot of research generated solutions which have implicated the data plane evolution. The evolution of data plane denotes a gradual deviation from original data plane architectures given by ForCES and OpenFlow specifications. A comprehensive review of SDN-related research was made to identify the key factors influencing the data plane evolution. Then, the correlation between treated problems and problem-solving approaches was established and shown in the correlation table. Afterwards, key limitations of ForCES and OpenFlow data plane architectures were identified, which set the conditions for selecting a particular problem-solving approach. Based on identified key limitations and using subjective metric, approaches to address the

problem of programmability and flexibility of SDN's data plane were generalized in four categories, as illustrated in the outer belt of the Figure 5:

A) data plane languages,
B) stateful data plane,
C) deeply programmable networks,
D) new data plane architectures and abstractions.

Open issues were identified by a critical review of generic problem-solving approaches in terms of flexibility and programmability, as shown in the middle belt of Figure 5. This establishes the ground for future research directions proposal (inner belt of the Figure 5) as follows:

- Development of platform-agnostic language for both description and programming of all processes in the data plane.
- Development of fully synchronized stateful data plane with support for state monitoring and management.
- Development of novel and platform-agnostic deeply programmable network architecture.

In closing of this paper, it is important to emphasize that the research surveyed in this paper did not provide the complete solution to recognized problems. Since simple extensions cannot solve problems of programmability and flexibility of the existing data plane architectures, we advocate the idea of creating an entirely new SDN's data plane architecture which will provide a high degree of flexibility for the upcoming network evolution, as illustrated in the center of Figure 5.

ACKNOWLEDGMENT

The authors would like to thank the Editor and anonymous reviewers that greatly helped to improve both the content and the readability of this paper. Mesud Hadzialic, deceased, was with the Department of Telecommunications, Faculty of Electrical Engineering, University of Sarajevo, Sarajevo, Bosnia and Herzegovina.

REFERENCES

[1] I. Leslie and D. McAuley, "Fairisle: An ATM network for the local area," in *Proc. Conf. Commun. Archit. Protocols*, New York, NY, USA, Sep. 1991, p. 327.

[2] J. E. van der Merwe, S. Rooney, L. Leslie, and S. Crosby, "The tempest-a practical framework for network programmability," *IEEE Netw.*, vol. 12, no. 3, pp. 20–28, May/Jun. 1998.

[3] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Commun. Mag.*, vol. 35, no. 1, pp. 80–86, Jan. 1997.

[4] A. T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente, and D. Villela, "A survey of programmable networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 2, pp. 7–23, Apr. 1999.

[5] 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.

[6] H. M. Khosravi and T. A. Anderson, *Requirements for Separation of IP Control and Forwarding*, document RFC3654, Dec. 2003, Accessed: Oct. 12, 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc3654.txt

[7] L. Yang, R. Dantu, T. A. Anderson, and R. Gopal, *Forwarding and Control Element Separation (ForCES) Framework*, document RFC3746, Apr. 2004, Accessed: Oct. 12, 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc3746.txt

[8] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[9] L. Dong, R. Gopal, and J. Halpern, *Forwarding and Control Element Separation (ForCES) Protocol Specification*, document RFC5810, Mar. 2010, Accessed: Oct. 12, 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc5810.txt

[10] Z. Wang, T. Tsou, J. Huang, X. Shi, and X. Yin, "Analysis of comparisons between OpenFlow and ForCES," in *Proc. Internet Draft*, Mar. 2012, pp. 1–13, Accessed: Oct. 12, 2018. [Online]. Available: https://tools.ietf.org/html/draft-wang-forces-compare-openflow-forces-01

[11] D. Kreutz, F. Ramos, P. E. Veríssimo, 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.

[12] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 27–51, 1st Quart., 2014.

[13] H. Farhady, L. HyunYong, and N. Akihiro, "Software-defined networking: A survey," *Comput. Netw.*, vol. 81, pp. 79–95, Apr. 2015.

[14] R. Masoudi and A. Ghaffari, "Software defined networks: A survey," *J. Netw. Comput. Appl.*, vol. 67, pp. 1–25, May 2016.

[15] J. H. Cox *et al.*, "Advancing software-defined networks: A survey," *IEEE Access*, vol. 5, pp. 25487–25526, 2017.

[16] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Comput. Netw.*, vol. 71, pp. 1–30, Oct. 2014.

[17] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "Research challenges for traffic engineering in software defined networks," *IEEE Netw.*, vol. 30, no. 3, pp. 52–58, May/Jun. 2016.

[18] M. Karakus and A. Durresi, "Quality of service (QoS) in software defined networking (SDN): A survey," *J. Netw. Comput. Appl.*, vol. 80, pp. 200–218, Feb. 2016.

[19] K. Benzekki, A. El Fergougui, and A. E. Elalaoui, "Software-defined networking (SDN): A survey," *Secur. Commun. Netw.*, vol. 9, no. 18, pp. 5803–5833, Dec. 2016.

[20] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti, "Energy efficiency in the future Internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 2, pp. 223–244, 2nd Quart., 2011.

[21] M. F. Tuysuz, Z. K. Ankarali, and D. Gözüpek, "A survey on energy efficiency in software defined networks," *Comput. Netw.*, vol. 113, pp. 188–204, Feb. 2017.

[22] M. Anan, A. Al-Fuqaha, N. Nasser, T.-Y. Mu, and H. Bustam, "Empowering networking research and experimentation through software-defined networking," *J. Netw. Comput. Appl.*, vol. 70, pp. 140–155, Jul. 2016.

[23] W. Kellerer, A. Basta, and A. Blenk, "Using a flexibility measure for network design space analysis of SDN and NFV," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2016, pp. 423–428.

[24] W. Kellerer *et al.*, "How to measure network flexibility? A proposal for evaluating softwarized networks," *IEEE Commun. Mag.*, vol. 56, no. 10, pp. 186–192, Oct. 2018.

[25] M. He, A. M. Alba, A. Basta, A. Blenk, and W. Kellerer, "Flexibility in softwarized networks: Classifications and research challenges," *IEEE Commun. Surveys Tuts.*, to be published.

[26] H. Farhad, H. Lee, and A. Nakao, "Data plane programmability in SDN," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, Oct. 2014, pp. 583–588.

[27] A. Nakao, "Research and development on network virtualization technologies in japan," *J. Nat. Inst. Inf. Commun. Technol.*, vol. 62, no. 2, pp. 25–32, 2015.

[28] N. Zilberman, P. M. Watts, C. Rotsos, and A. W. Moore, "Reconfigurable network systems and software-defined networking," *Proc. IEEE*, vol. 103, no. 7, pp. 1102–1124, Jul. 2015.

[29] 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.

[30] R. Bifulco and G. Rétvári, "A survey on the programmable data plane: Abstractions architectures and open problems," in *Proc. IEEE HPSR*, Jun. 2018, pp. 1–7.

[31] H.-H. Cho, C.-F. Lai, T. K. Shih, and H.-C. Chao, "Integration of SDR and SDN for 5G," *IEEE Access*, vol. 2, pp. 1196–1204, 2014.

[32] 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.
[33] S. Sun, M. Kadoch, L. Gong, and B. Rong, "Integrating network function virtualization with SDR and SDN for 4G/5G networks," *IEEE Netw.*, vol. 29, no. 3, pp. 54–59, May/Jun. 2015.
[34] J. M. Halpern and J. H. Salim, *Forwarding and Control Element Separation (ForCES) Forwarding Element Model*, document RFC5812, Mar. 2010, Accessed: Oct. 12, 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc5812.txt
[35] L. Dong, F. Jia, and W. Wang, "Definition and implementation of logical function blocks compliant to ForCES specification," in *Proc. 15th IEEE Int. Conf. Netw.*, Nov. 2007, pp. 531–536.
[36] P. L. G. Ramírez, J. Lloret, S. M. Cordero, and L. C. T. Arboleda, "Design and implementation of ForCES protocol," *Netw. Protocols Algorithms*, vol. 9, nos. 1–2, pp. 1–27, 2017.
[37] J. Rong, H. Xiong-Xiong, and Z. Guang-Xin, "A composition method of logical function block chain," *Int. J. Future Gener. Commun. Netw.*, vol. 9, no. 3, pp. 57–68, Mar. 2016.
[38] E. Haleplidis *et al.*, "Network programmability with ForCES," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1423–1440, 3rd Quart., 2015.
[39] C. Schuba, J. Goldschmidt, K. Kalajan, and M. F. Speer, "Integrated network service processing using programmable network devices," Sun Microsystems, Mountain View, CA, USA, Tech. Rep. 138, 2005.
[40] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 1–12, 2007.
[41] E. Haleplidis, S. Denazis, O. Koufopavlou, J. H. Salim, and J. Halpern, "Software-defined networking: Experimenting with the control to forwarding plane interface," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 91–96.
[42] D. A. Broniatowski and J. Moses, "Measuring flexibility, descriptive complexity, and rework potential in generic system architectures," *Syst. Eng.*, vol. 19, no. 3, pp. 207–221, May 2016.
[43] R. Nilchiani and D. E. Hastings, "Measuring the value of flexibility in space systems: A six-element framework," *Syst. Eng.*, vol. 10, no. 1, pp. 26–44.
[44] D. A. Broniatowski, "Flexibility due to abstraction and decomposition," *Syst. Eng.*, vol. 20, no. 2, pp. 98–117, Mar. 2017.
[45] M. Omer, R. Nilchiani, and A. Mostashari, "Measuring the resilience of the trans-oceanic telecommunication cable system," *IEEE Syst. J.*, vol. 3, no. 3, pp. 295–303, Sep. 2009.
[46] C. W. Zobel, "Representing perceived tradeoffs in defining disaster resilience," *Decis. Support Syst.*, vol. 50, no. 2, pp. 394–403, Jan. 2011.
[47] R. Filippini and A. Silva, "A modeling framework for the resilience analysis of networked systems-of-systems based on functional dependencies," *Rel. Eng. Syst. Saf.*, vol. 125, pp. 82–91, May 2014.
[48] M. Casado, G. Watson, and N. McKeown, "Reconfigurable networking hardware: A classroom tool," in *Proc. 13th Symp. High Perform. Interconnects (HOTI)*, Aug. 2005, pp. 151–157.
[49] M. Casado, G. Watson, and N. McKeown, "Teaching networking hardware," in *Proc. 10th Annu. SIGCSE Conf. Innov. Technol. Comput. Sci. Educ.*, New York, NY, USA, Jun. 2005, pp. 208–212.
[50] G. Watson, N. McKeown, and M. Casado, "NetFPGA: A tool for network research and education," in *Proc. 2nd Workshop Architectural Res. FPGA Platforms (WARFP)*, vol. 3, Feb. 2006, pp. 1–4.
[51] J. W. Lockwood *et al.*, "NetFPGA—An open platform for gigabit-rate network switching and routing," in *Proc. IEEE Int. Conf. Microelectron. Syst. Education (MSE)*, Jun. 2007, pp. 160–161.
[52] J. Luo, J. Pettit, M. Casado, J. Lockwood, and N. McKeown, "Prototyping fast, simple, secure switches for etha," in *Proc. 15th Annu. IEEE Symp. High-Perform. Interconnects (HOTI)*, Aug. 2007, pp. 73–82.
[53] M. Blott, J. Ellithorpe, N. McKeown, K. Vissers, and H. Zeng, "FPGA research design platform fuels network advances," *Xilinx Xcell J.*, vol. 4, no. 73, pp. 24–29, Sep. 2010.
[54] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 Gbps as research commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, Sep./Oct. 2014.
[55] J. Cao, X. Zheng, L. Sun, and J. Jin, "The development status and trend of NetFPGA," in *Proc. Int. Conf. Netw. Inf. Syst. Comput.*, Jan. 2015, pp. 101–105.
[56] P. Varga, L. Kovács, T. Tóthfalusi, and P. Orosz, "C-GEP: 100 Gbit/s capable, FPGA-based, reconfigurable networking equipment," in *Proc. IEEE 16th Int. Conf. High Perform. Switching Routing (HPSR)*, Jul. 2015, pp. 1–6.
[57] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an OpenFlow switch on the NetFPGA platform," in *Proc. 4th ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, New York, NY, USA, Nov. 2008, pp. 1–9.
[58] G. Gibb, J. W. Lockwood, J. Naous, P. Hartke, and N. McKeown, "NetFPGA—An open platform for teaching how to build gigabit-rate network switches and routers," *IEEE Trans. Educ.*, vol. 51, no. 3, pp. 364–369, Aug. 2008.
[59] J. Naous, G. Gibb, S. Bolouki, and N. McKeown, "NetFPGA: Reusable router architecture for experimental research," in *Proc. ACM Workshop Program. Routers Extensible Services Tomorrow*, New York, NY, USA, Aug. 2008, pp. 1–7.
[60] A. Khan and N. Dave, "Enabling hardware exploration in software-defined networking: A flexible, portable OpenFlow switch," in *Proc. IEEE 21st Annu. Int. Symp. Field-Program. Custom Comput. Mach.*, Apr. 2013, pp. 145–148.
[61] G. Gibb and N. McKeown, "OpenPipes: Making distributed hardware systems easier," in *Proc. Int. Conf. Field-Program. Technol.*, Dec. 2010, pp. 381–384.
[62] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf, "Enabling FPGAs in hyperscale data centers," in *Proc. IEEE 12th Intl Conf. Ubiquitous Intell. Comput., IEEE 12th Intl Conf. Autonomic Trusted Comput., IEEE 15th Intl Conf. Scalable Comput. Commun. Associated Workshops (UIC-ATC-ScalCom)*, Aug. 2015, pp. 1078–1086.
[63] A. Bitar, M. S. Abdelfattah, and V. Betz, "Bringing programmability to the data plane: Packet processing with a NoC-enhanced FPGA," in *Proc. Int. Conf. Field Program. Technol. (FPT)*, Dec. 2015, pp. 24–31.
[64] G. Antichi, M. Shahbaz, S. Giordano, and A. Moore, "From 1G to 10G: Code reuse in action," in *Proc. 1st Ed. Workshop High Perform. Program. Netw.*, New York, NY, USA, Jun. 2013, pp. 31–38.
[65] C. Hu, J. Yang, H. Zhao, and J. Lu, "Design of all programable innovation platform for software defined networking," in *Proc. Presented Part Open Netw. Summit (ONS)*. Santa Clara, CA, USA, 2014, pp. 1–2.
[66] S. Zhou, W. Jiang, and V. Prasanna, "A programmable and scalable OpenFlow switch using heterogeneous SoC platforms," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, New York, NY, USA, Aug. 2014, pp. 239–240.
[67] G. Lu *et al.*, "ServerSwitch: A programmable and high performance platform for data center networks," in *Proc. 8th USENIX Symp. Netw. Syst. Design Implement.*, Berkeley, CA, USA, Mar./Apr. 2011, pp. 15–28.
[68] W. Wang *et al.*, "Design and implementation of an open programmable router compliant to IETF ForCES specifications," in *Proc. 6th Int. Conf. Netw. (ICN)*, Apr. 2007, p. 82.
[69] A. Rostami, T. Jungel, A. Koepsel, H. Woesner, and A. Wolisz, "ORAN: OpenFlow routers for academic networks," in *Proc. IEEE 13th Int. Conf. High Perform. Switching Routing*, Jun. 2012, pp. 216–222.
[70] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comp. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000.
[71] M. Handley, O. Hodson, and E. Kohler, "XORP: An open platform for network research," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 53–57, Jan. 2003.
[72] M. Dobrescu *et al.*, "RouteBricks: Exploiting parallelism to scale software routers," in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Princ.*, New York, NY, USA, Oct. 2009, pp. 15–28.
[73] Y. Mundada, R. Sherwood, and N. Feamster, "An OpenFlow switch element for click," in *Proc. Symp. Click Modular Router*, Jan. 2009, p. 1.
[74] E. L. Fernandes and C. E. Rothenberg, "OpenFlow 1.3 software switch," in *Proc. Salao de Ferramentas do 32nd Simp. Brasileiro de Redes de Computadores e Sistemas Distribuidos (SBRC)*, 2014, pp. 1021–1028.
[75] F. Risso and M. Baldi, "NetPDL: An extensible XML-based language for packet header description," *Comput. Netw.*, vol. 50, no. 5, pp. 688–706, Apr. 2006.
[76] H. Kim, J. Kim, and Y.-B. Ko, "Developing a cost-effective OpenFlow testbed for small-scale software defined networking," in *Proc. 16th Int. Conf. Adv. Commun. Technol.*, Feb. 2014, pp. 758–761.
[77] (2009). *OpenFlow Reference Implementation*. Accessed: Oct. 12, 2018. [Online]. Available: https://github.com/mininet/openflow
[78] (2013). *Indigo*. Accessed: Oct. 12, 2018. [Online]. Available: http://www.projectfloodlight.org/indigo/

[79] (2013). *Pantou*. Accessed: Oct. 12, 2018. [Online]. Available: https://github.com/CPqD/openflow-openwrt
[80] (2012). *OpenFaucet*. Accessed: Oct. 12, 2018. [Online]. Available: https://github.com/rlenglet/openfaucet
[81] (2013). *OpenFlow Java*. Accessed: Oct. 12, 2018. [Online]. Available: https://bitbucket.org/openflowj/openflowj/
[82] (2011). *Oflib-Node*. Accessed: Oct. 12, 2018. [Online]. Available: https://github.com/TrafficLab/oflib-node
[83] E. L. Fernandes, G. Antichi, I. Castro, and S. Uhlig, "An SDN-inspired model for faster network experimentation," in *Proc. ACM SIGSIM Conf. Princ. Adv. Discrete Simulation*, New York, NY, USA, May 2018, pp. 29–32.
[84] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw.*, New York, NY, USA, Oct. 2010, Art. no. 19.
[85] M. Ahmed, F. Huici, and A. Jahanpanah, "Enabling dynamic network processing with clickOS," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, New York, NY, USA, Aug. 2012, pp. 293–294.
[86] L. Rizzo, "Netmap: A novel framework for fast packet I/O," in *Proc. USENIX Conf. Annu. Tech. Conf.*, Berkeley, CA, USA, Jun. 2012, p. 9.
[87] M. Honda, F. Huici, G. Lettieri, and L. Rizzo, "mSwitch: A highly-scalable, modular software switch," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, New York, NY, USA, Jun. 2015, Art. no. 1.
[88] B. Pfaff *et al.*, "The design and implementation of open vSwitch," in *Proc. 12th USENIX Conf. Netw. Syst. Design Implement.*, Berkeley, CA, USA, May 2015, pp. 117–130.
[89] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: A GPU-accelerated software router," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 195–206, 2010.
[90] D. Zhou, B. Fan, H. Lim, M. Kaminsky, and D. G. Andersen, "Scalable, high performance Ethernet forwarding with CuckooSwitch," in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol.*, New York, NY, USA, Dec. 2013, pp. 97–108.
[91] Intel. (2010). *Data Plane Development Kit*. Accessed: Oct. 12, 2018. [Online]. Available: https://software.intel.com/en-us/networking/dpdk
[92] G. Pongrácz, L. Molnár, and Z. L. Kis, "Removing roadblocks from SDN: openflow software switch performance on intel DPDK," in *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, Oct. 2013, pp. 62–67.
[93] J. Keinänen, P. Jokela, and K. Slavov, "Implementing zFilter based forwarding node on a NetFPGA," in *Proc. NetFPGA Developers Workshop* , Aug. 2009, pp. 1–8.
[94] G. Antichi, A. Di Pietro, S. Giordano, G. Procissi, and D. Ficara, "Design and development of an OpenFlow compliant smart gigabit switch," in *Proc. IEEE Global Telecommun. Conf.-GLOBECOM*, Dec. 2011, pp. 1–5.
[95] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," *Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.
[96] O. El Ferkouss *et al.*, "A 100Gig network processor platform for OpenFlow," in *Proc. 7th Int. Conf. Netw. Service Manage.*, Oct. 2011, pp. 1–4.
[97] Y. R. Qu, S. Zhou, and V. K. Prasanna, "High-performance architecture for dynamically updatable packet classification on FPGA," in *Proc. Archit. Netw. Commun. Syst.*, Oct. 2013, pp. 125–136.
[98] M. Martinello, M. R. N. Ribeiro, R. E. Z. de Oliveira, and R. de Angelis Vitoi, "Keyflow: A prototype for evolving SDN toward core network fabrics," *IEEE Netw.*, vol. 28, no. 2, pp. 12–19, Mar. 2014.
[99] K. G. Pérez, X. Yang, S. Scott-Hayward, and S. Sezer, "A configurable packet classification architecture for software-defined networking," in *Proc. 27th IEEE Int. Syst.-on-Chip Conf. (SOCC)*, Sep. 2014, pp. 353–358.
[100] K. G. Pérez, X. Yang, S. Scott-Hayward, and S. Sezer, "Optimized packet classification for software-defined networking," in *Proc. IEEE Int. Conf. on Commun. (ICC)*, Jun. 2014, pp. 859–864.
[101] B. Yan, Y. Xu, H. Xing, K. Xi, and H. J. Chao, "CAB: A reactive wildcard rule caching system for software-defined networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, New York, NY, USA, Aug. 2014, pp. 163–168.
[102] Y. Li, D. Zhang, K. Huang, D. He, and W. Long, "A memory-efficient parallel routing lookup model with fast updates," *Comput. Commun.*, vol. 38, pp. 60–71, Feb. 2014.
[103] A. Kalyaev, I. Korovin, M. Khisamutdinov, G. Schaefer, and M. A. R. Ahad, "A hardware approach for organisation of software defined network switches based on FPGA," in *Proc. Int. Conf. Inform., Electron. Vis. (ICIEV)*, Jun. 2015, pp. 1–4.
[104] M. Ciesla, V. Sivaraman, and A. Seneviratne, "URL extraction on the NetFPGA reference router," in *Proc. NetFPGA Developers Workshop*, Aug. 2009, pp. 1–6.
[105] Y. Luo, P. Cascon, E. Murray, and J. Ortega, "Accelerating OpenFlow switching with network processors," in *Proc. 5th ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, New York, NY, USA, Oct. 2009, pp. 70–71.
[106] K. K. Ram, J. Mudigonda, A. L. Cox, S. Rixner, P. Ranganathan, and J. R. Santos, "sNICh: Efficient last hop networking in the data center," in *Proc. 6th ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, New York, NY, USA, Oct. 2010, Art. no. 26.
[107] V. Tanyingyong, M. Hidell, and P. Sjödin, "Improving PC-based OpenFlow switching performance," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, New York, NY, USA, Oct. 2010, pp. 1–2.
[108] V. Tanyingyong, M. Hidell, and P. Sjödin, "Using hardware classification to improve PC-based OpenFlow switching," in *Proc. IEEE 12th Int. Conf. High Perform. Switching Routing*, Jul. 2011, pp. 215–221.
[109] S. Gao, S. Shimizu, S. Okamoto, and N. Yamanaka, "A high-speed routing engine for software defined network," *J. Sel. Areas Telecommun.*, vol. 2, pp. 1–7, Aug. 2012.
[110] G. Lu, R. Miao, Y. Xiong, and C. Guo, "Using CPU as a traffic co-processing unit in commodity switches," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, New York, NY, USA, Aug. 2012, pp. 31–36.
[111] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite CacheFlow in software-defined networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, New York, NY, USA, Aug. 2014, pp. 175–180.
[112] R. D. Vencioneck, G. Vassoler, M. Martinello, M. R. N. Ribeiro, and C. Marcondes, "FlexForward: Enabling an SDN manageable forwarding engine in Open vSwitch," in *Proc. 10th Int. Conf. Netw. Service Manage. (CNSM) Workshop*, Nov. 2014, pp. 296–299.
[113] H. T. Dang, D. Sciascia, M. Canini, F. Pedone, and R. Soulé, "NetPaxos: Consensus at network speed," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, New York, NY, USA, Jun. 2015, Art. no. 5.
[114] R. Bifulco and A. Matsiuk, "Towards scalable SDN switches: Enabling faster flow table entries installation," in *Proc. ACM Conf. Special Interest Group Data Commun.*, New York, NY, USA, 2015, pp. 343–344.
[115] D. Sanvito, D. Moro, and A. Capone, "Towards traffic classification offloading to stateful SDN data planes," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Jul. 2017, pp. 1–4.
[116] L. N. Bhuyan and D. P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Trans. Comput.*, vol. 33, no. 4, pp. 323–333, Apr. 1984.
[117] K. Kannan and S. Banerjee, *Compact TCAM: Flow Entry Compaction in TCAM for Power Aware SDN*. Berlin, Germany: Springer, 2013, pp. 439–444.
[118] S. Banerjee and K. Kannan, "Tag-In-Tag: Efficient flow table management in SDN switches," in *Proc. 10th Int. Conf. Netw. Service Manage. (CNSM) Workshop*, Nov. 2014, pp. 109–117.
[119] P. T. Congdon, P. Mohapatra, M. Farrens, and V. Akella, "Simultaneously reducing latency and power consumption in OpenFlow switches," *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 1007–1020, Jun. 2014.
[120] T. H. Vu *et al.*, "Power aware OpenFlow switch extension for energy saving in data centers," in *Proc. Int. Conf. Adv. Technol. Commun.*, Oct. 2012, pp. 309–313.
[121] R. Bolla *et al.*, "The green abstraction layer: A standard power-management interface for next-generation network devices," *IEEE Internet Comput.*, vol. 17, no. 2, pp. 82–86, Mar. 2013.
[122] N.-S. Ko, H. Heo, J.-D. Park, and H.-S. Park, "OpenQFlow: Scalable OpenFlow with flow-based QoS," *IEICE Trans. Commun.*, vol. E96.B, no. 2, pp. 479–488, 2013.
[123] B. Sonkoly *et al.*, "On QoS support to Ofelia and OpenFlow," in *Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 109–113.
[124] N. B. Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, and L. Veltri, "An OpenFlow-based testbed for information centric networking," in *Proc. Future Netw. Mobile Summit (FutureNetw)*, Jul. 2012, pp. 1–9.
[125] (Jun. 2012). *OpenFlow Configuration and Managementgement Protocol OF-CONFIG 1.0*. Accessed: Oct. 12, 2018. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2013/02/of-config1dot0-final.pdf

[126] R. Enns, M. Björklund, A. Bierman, and J. Schönwälder, *Network Configuration Protocol (NETCONF)*, document RFC6241, Jun. 2011, Accessed: Oct. 12, 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc6241.txt
[127] S. Jain *et al.*, "olzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, Oct. 2013.
[128] D. Palma *et al.*, "The QueuePusher: Enabling queue management in OpenFlow," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, Sep. 2014, pp. 125–126.
[129] C. Caba and J. Soler, "APIs for QoS configuration in software defined networks," in *Proc. 1st IEEE Conf. Netw. Softwarization (NetSoft)*, Apr. 2015, pp. 1–5.
[130] A. Sivaraman, K. Winstein, S. Subramanian, and H. Balakrishnan, "No silver bullet: Extending SDN to the data plane," in *Proc. 12th ACM Workshop Hot Topics Netw.*, New York, NY, USA, Nov. 2013, Art. no. 19.
[131] T. H. Szymanski and M. Rezaee, "An FPGA controller for deterministic guaranteed-rate optical packet switching," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2015, pp. 1177–1183.
[132] W. Wang, Q. Qi, X. Gong, Y. Hu, and X. Que, "Autonomic QoS management mechanism in software defined network," *China Commun.*, vol. 11, no. 7, pp. 13–23, Jul. 2014.
[133] W. Wang, Y. Tian, X. Gong, Q. Qi, and Y. Hu, "Software defined autonomic QoS model for future Internet," *J. Syst. Softw.*, vol. 110, pp. 122–135, Dec. 2015.
[134] P. Wette and H. Karl, "Which flows are hiding behind my wildcard rule?: Adding packet sampling to openflow," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 541–542, Oct. 2013.
[135] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, "I know what your packet did last hop: Using packet histories to troubleshoot networks," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*. Seattle, WA, USA, Apr. 2014, pp. 71–85.
[136] Q. Zuo, M. Chen, K. Ding, and B. Xu, "On generality of the data plane and scalability of the control plane in software-defined networking," *China Commun.*, vol. 11, no. 2, pp. 55–64, Feb. 2014.
[137] M. Bonola, G. Bianchi, G. Picierro, S. Pontarelli, and M. Monaci, "StreaMon: A data-plane programming abstraction for software-defined stream monitoring," *IEEE Trans. Dependable Secure Comput.*, vol. 14, no. 6, pp. 664–678, Nov./Dec. 2017.
[138] J. C. Mogul and P. Congdon, "Hey, you darned counters!: Get off my ASIC!" in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, New York, NY, USA, Aug. 2012, pp. 25–30.
[139] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Proc. 10th USENIX Conf. Netw. Syst. Design Implement.*, Berkeley, CA, USA, Apr. 2013, pp. 29–42.
[140] V. Jeyakumar, M. Alizadeh, C. Kim, and D. Mazières, "Tiny packet programs for low-latency network control and monitoring," in *Proc. 12th ACM Workshop Hot Topics Netw.*, New York, NY, USA, Nov. 2013, Art. no. 8.
[141] Y. Choi, "Implementation of content-oriented networking architecture (CONA): A focus on DDoS countermeasure," in *Proc. 1st Eur. NetFPGA Developers Workshop*, Sep. 2010, pp. 1–6.
[142] K. Benzekki, A. El Fergougui, and A. El Belrhiti El Alaoui, "Devolving IEEE 802.1X authentication capability to data plane in software-defined networking (SDN) architecture," *Secur. Commun. Netw.*, vol. 9, no. 17, pp. 4369–4377, Nov. 2016.
[143] A. Fiessler, S. Hager, B. Scheuermann, and A. W. Moore, "HyPaFilter—A versatile hybrid FPGA packet filter," in *Proc. Symp. Archit. Netw. Commun. Syst.*, Mar. 2016, pp. 25–36.
[144] W. Han *et al.*, "State-aware network access management for software-defined networks," in *Proc. 21st ACM Symp. Access Control Models Technol.*, New York, NY, USA, Jun. 2016, pp. 1–11.
[145] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, and P. Sköldström, "Scalable fault management for OpenFlow," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2012, pp. 6606–6610.
[146] A. Capone, C. Cascone, A. Q. T. Nguyen, and B. Sansò, "Detour planning for fast and reliable failure recovery in SDN with OpenState," in *Proc. 11th Int. Conf. Design Reliable Commun. Netw. (DRCN)*, Mar. 2015, pp. 25–32.
[147] C. Cascone, D. Sanvito, L. Pollini, A. Capone, and B. Sansò, "Fast failure detection and recovery in SDN with stateful data plane," *Int. J. Netw. Manage.*, vol. 27, no. 2, p. e1957, Mar./Apr. 2017.
[148] L. Petrucci, M. Bonola, S. Pontarelli, G. Bianchi, and R. Bifulco, "Implementing iptables using a programmable stateful data plane abstraction: Demo," in *Proc. Symp. SDN Res.*, New York, NY, USA, Apr. 2017, pp. 193–194.
[149] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "Openstate: Programming platform-independent stateful openflow applications inside the switch," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, 2014.
[150] J. Kempf *et al.*, "OpenFlow MPLS and the open source label switched router," in *Proc. 23rd Int. Teletraffic Congr. (ITC)*, Sep. 2011, pp. 8–14.
[151] M. Shirazipour, W. John, J. Kempf, H. Green, and M. Tatipamula, "Realizing packet-optical integration with SDN and OpenFlow 1.1 extensions," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2012, pp. 6633–6637.
[152] A. Sadasivarao *et al.*, "Open Transport Switch: A software defined networking architecture for transport networks," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, New York, NY, USA, Aug. 2013, pp. 115–120.
[153] N. Amaya *et al.*, "First demonstration of software defined networking (SDN) over space division multiplexing (SDM) optical networks," in *Proc. 39th Eur. Conf. Exhib. Opt. Commun. (ECOC)*, Sep. 2013, pp. 1–3.
[154] Y. Yan, Y. Shu, G. M. Saridis, B. R. Rofoee, G. Zervas, and D. Simeonidou, "FPGA-based optical programmable switch and interface card for disaggregated OPS/OCS data centre networks," in *Proc. Eur. Conf. Opt. Commun. (ECOC)*, Sep./Oct. 2015, pp. 1–3.
[155] W. Miao *et al.*, "SDN-enabled OPS with QoS guarantee for reconfigurable virtual data center networks," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 7, no. 7, pp. 634–643, Jul. 2015.
[156] K. Kondepu, A. Sgambelluri, L. Valcarenghi, F. Cugini, and P. Castoldi, "An SDN-based integration of green TWDM-PONs and metro networks preserving end-to-end delay," in *Proc. Opt. Fiber Commun. Conf. Exhib. (OFC)*, Mar. 2015, pp. 1–3.
[157] P. Bakopoulos *et al.*, "NEPHELE: An end-to-end scalable and dynamically reconfigurable optical architecture for application-aware SDN cloud data centers," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 178–188, Feb. 2018.
[158] W. John, A. Kern, M. Kind, P. Skoldstrom, D. Staessens, and H. Woesner, "Splitarchitecture: SDN for the carrier domain," *IEEE Commun. Mag.*, vol. 52, no. 10, pp. 146–152, Oct. 2014.
[159] G. M. Saridis *et al.*, "LIGHTNESS: A deeply-programmable SDN-enabled data centre network with OCS/OPS multicast/unicast switchover," in *Proc. Eur. Conf. Opt. Commun. (ECOC)*, Sep. 2015, pp. 1–3.
[160] J.-P. Elbers and A. Autenrieth, "From static to software-defined optical networks," in *Proc. 16th Int. Conf. Opt. Netw. Design Modelling (ONDM)*, Apr. 2012, pp. 1–4.
[161] D. Zhang, X. Song, C. Chen, H. Guo, J. Wu, and Y. Xia, "Software defined synergistic IP+optical resilient transport networks [Invited]," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 7, no. 2, pp. A209–A217, Feb. 2015.
[162] Y. Xiong, Z. Li, B. Zhou, and X. Dong, "Cross-layer shared protection strategy towards data plane in software defined optical networks," *Opt. Commun.*, vol. 412, pp. 66–73, Apr. 2018.
[163] B. Belter *et al.*, "Hardware abstraction layer as an SDN-enabler for non-OpenFlow network equipment," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, Sep. 2014, pp. 117–118.
[164] L. Ogrodowczyk *et al.*, "Hardware abstraction layer for non-OpenFlow capable devices," in *Proc. TERENA Netw. Conf.*, May 2014, pp. 1–15.
[165] D. Parniewicz *et al.*, "Design and implementation of an OpenFlow hardware abstraction layer," in *Proc. ACM SIGCOMM Workshop Distrib. Cloud Comput.*, New York, NY, USA, Aug. 2014, pp. 71–76.
[166] R. G. Clegg, J. Spencer, R. Landa, M. Thakur, J. Mitchell, and M. Rio, "Pushing software defined networking to the access," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, Sep. 2014, pp. 31–36.
[167] V. Fuentes, J. Matias, A. Mendiola, M. Huarte, J. Unzilla, and E. Jacob, "Integrating complex legacy systems under OpenFlow control: The DOCSIS use case," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, Sep. 2014, pp. 37–42.
[168] K. Kondepu *et al.*, "Fully SDN-enabled all-optical architecture for data center virtualization with time and space multiplexing," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 10, no. 7, pp. 90–101, Jul. 2018.
[169] X. Ge *et al.*, "OpenANFV: Accelerating network function virtualization with a consolidated framework in openstack," in *Proc. ACM Conf. SIGCOMM*, New York, NY, USA, Aug. 2014, pp. 353–354.

[170] C. Kachris, G. C. Sirakoulis, and D. Soudris. (2014). "Network function virtualization based on FPGAs: A framework for all-programmable network devices." [Online]. Available: https://arxiv.org/abs/1406.0309

[171] D. Hancock and J. Van Der Merwe, *HyPer4: Using P4 to Virtualize the Programmable Data Plane*. New York, NY, USA: Association for Computing Machinery, 2016, pp. 35–49.

[172] R. Jin, X. He, and M. Gao, "A method of ForCES virtualization," *Int. J. Future Gener. Commun. Netw.*, vol. 9, no. 1, pp. 271–284, Jan. 2016.

[173] D. Firestone, "VFP: A virtual switch platform for host SDN in the public cloud," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Boston, MA, USA, 2017, pp. 315–328.

[174] N. Oguchi and M. Sekiya, "Virtual data planes for easy creation and operation of end-to-end virtual networks," in *Proc. 25th Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM)*, Sep. 2017, pp. 1–6.

[175] B. Sonkoly, R. Szabo, D. Jocha, J. Czentye, M. Kind, and F.-J. Westphal, "UNIFYing cloud and carrier network resources: An architectural view," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–7.

[176] M. Szabo, A. Majdan, G. Pongracz, L. Toka, and B. Sonkoly, "Making the data plane ready for NFV: An effective way of handling resources," in *Proc. SIGCOMM Posters Demos*, New York, NY, USA, Aug. 2017, pp. 97–99.

[177] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

[178] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *Proc. ACM SIGCOMM Conference SIGCOMM*, New York, NY, USA, Aug. 2013, pp. 99–110.

[179] L. Jose, L. Yan, G. Varghese, and N. McKeown, "Compiling packet programs to reconfigurable switches," in *Proc. 12th USENIX Conf. Netw. Syst. Design Implement.*, Berkeley, CA, USA, May 2015, pp. 103–115.

[180] A. Sivaraman, C. Kim, R. Krishnamoorthy, A. Dixit, and M. Budiu, "DC.p4: Programming the forwarding plane of a data-center switch," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, New York, NY, USA, Jun. 2015, Art. no. 2.

[181] M. Shahbaz *et al.*, "PISCES: A programmable, protocol-independent software switch," in *Proc. ACM SIGCOMM Conf.*, New York, NY, USA, Aug. 2016, pp. 525–538.

[182] H. Wang, K. S. Lee, V. Shrivastav, and H. Weatherspoon, "P4FPGA: High level synthesis for networking," in *Proc. ACM SIGCOMM Workshop Netw. Program. Lang. (NetPL)*, Brazil, South America, 2016
[182] is a Short Paper: ACM SIGCOMM Workshop on Networking and Programming Languages (NetPL 2016), Brazil 2016

[183] H. Wang *et al.*, "P4FPGA: A rapid prototyping framework for P4," in *Proc. Symp. SDN Res.*, New York, NY, USA, Apr. 2017, pp. 122–135.

[184] L. Wirbel, "Xilinx SDNet: A new way to specify network hardware," The Linley Group, Mountain View, CA, USA, White Paper no., 2014.

[185] M. Shahbaz and N. Feamster, "The case for an intermediate representation for programmable data planes," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, New York, NY, USA, Jun. 2015, Art. no. 3.

[186] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, New York, NY, USA, Aug. 2013, pp. 127–132.

[187] A. Sivaraman *et al.*, "Packet transactions: High-level programming for line-rate switches," in *Proc. ACM SIGCOMM Conf.*, New York, NY, USA, Aug. 2016, pp. 15–28.

[188] S. Zhu, J. Bi, and C. Sun, "SFA: Stateful forwarding abstraction in SDN data plane," in *Proc. Presented as Part Open Netw. Summit (ONS)*. Santa Clara, CA, USA, Mar. 2014, pp. 1–2.

[189] S. Zhu, J. Bi, C. Sun, C. Wu, and H. Hu, "SDPA: Enhancing stateful forwarding for software-defined networking," in *Proc. IEEE 23rd Int. Conf. Netw. Protocols (ICNP)*, Nov. 2015, pp. 323–333.

[190] G. Bianchi, M. Bonola, A. Capone, C. Cascone, and S. Pontarelli. (Aug. 2014). "Towards wire-speed platform-agnostic control of OpenFlow switches." [Online]. Available: https://arxiv.org/abs/1409.0242

[191] G. Bianchi, M. Bonola, S. Pontarelli, D. Sanvito, A. Capone, and C. Cascone. (2016). "Open Packet Processor: A programmable architecture for wire speed platform-independent stateful in-network processing." [Online]. Available: https://arxiv.org/abs/1605.01977

[192] M. Bonola, R. Bifulco, L. Petrucci, S. Pontarelli, A. Tulumello, and G. Bianchi, "Implementing advanced network functions for datacenters with stateful programmable data planes," in *Proc. IEEE Int. Symp. Local Metrop. Area Netw. (LANMAN)*, Jun. 2017, pp. 1–6.

[193] A. Nakao, "VNode: A deeply programmable network testbed through network virtualization," *3rd IEICE Tech. Committee Netw. Virtualization*, vol. 112, no. 230, pp. 103–108, Mar. 2012.

[194] Y. Kanada, K. Shiraishi, and A. Nakao, "High-performance network accommodation and intra-slice switching using a type of virtualization node," in *Proc. INFOCOM*, 2012, pp. 1–7.

[195] A. Nakao, "Deeply programmable network; Emerging technologies for network virtualization and software defined network (SDN)," in *Proc. ITU Kaleidoscope, Building Sustain. Communities*, Apr. 2013, p. 1.

[196] K. Yamada, Y. Kanada, K. Amemiya, A. Nakao, and Y. Saida, "VNode infrastucture enhancement—Deeply programmable network virtualization," in *Proc. 21st Asia–Pacific Conf. Commun. (APCC)*, Oct. 2015, pp. 244–249.

[197] A. Nakao, "Software-defined data plane enhancing SDN and NFV," *IEICE Trans. Commun.*, vol. E98.B, no. 1, pp. 12–19, 2015.

[198] K. Yamada, A. Nakao, Y. Kanada, Y. Saida, K. Amemiya, and Y. Minami, "Design and deployment of enhanced VNode infrastructure—Deeply programmable network virtualization," *IEICE Trans. Commun.*, vol. 99, no. 8, pp. 1629–1637, 2016.

[199] Y. Minami and K. Yamada, "Novel applications and experiments on programmable infrastructures," in *Proc. 24th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2015, pp. 1–6.

[200] A. Nakao, P. Du, and T. Iwai, "Application specific slicing for MVNO through software-defined data plane enhancing SDN," *IEICE Trans. Commun.*, vol. E98.B, no. 11, pp. 2111–2120, 2015.

[201] P. Du, P. Putra, S. Yamamoto, and A. Nakao, "A context-aware IoT architecture through software-defined data plane," in *Proc. IEEE Region Symp. (TENSYMP)*, May 2016, pp. 315–320.

[202] A. Nakao *et al.*, "End-to-end network slicing for 5G mobile networks," *J. Inf. Process.*, vol. 25, pp. 153–163, 2017.

[203] S. Ando and A. Nakao, "L7 packet switch: Packet switch applying regular expression to packet payload," in *Proc. IEEE Int. Workshop Tech. Committee Commun. Qual. Rel. (CQR)*, May 2014, pp. 1–6.

[204] H. Farhady and A. Nakao, "TagFlow: Efficient flow classification in SDN," *IEICE Trans. Commun.*, vol. 97, no. 11, pp. 2302–2310, 2014.

[205] H. Farhadi, P. Du, and A. Nakao, "User-defined actions for SDN," in *Proc. 9th Int. Conf. Future Internet Technol.*, New York, NY, USA, Jun. 2014, Art. no. 3.

[206] S. Nirasawa, M. Hara, S. Yamaguchi, M. Oguchi, A. Nakao, and S. Yamamoto, "Application performance improvement with application aware DPN switches," in *Proc. 18th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Oct. 2016, pp. 1–4.

[207] S. Nirasawa, M. Hara, A. Nakao, M. Oguchi, S. Yamamoto, and S. Yamaguchi, "Network application performance improvement with deeply programmable switch," in *Proc. 13th Int. Conf. Mobile Ubiquitous Syst., Comput. Netw. Services*, ser. MOBIQUITOUS 2016.New York, NY, USA, Nov. 2016, pp. 263–267.

[208] S. Nirasawa, A. Nakao, S. Yamamoto, M. Hara, M. Oguchi, and S. Yamaguchi, "Application switch using DPN for improving TCP based data center applications," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 995–998.

[209] M. B. Anwer, M. Motiwala, M. B. Tariq, and N. Feamster, "SwitchBlade: A platform for rapid deployment of network protocols on programmable hardware," *ACM SIGCOMM Comput. Commu. Rev.*, vol. 40, no. 4, pp. 183–194, Oct. 2010.

[210] R. Narayanan *et al.*, "Macroflows and microflows: Enabling rapid network innovation through a split SDN data plane," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 79–84.

[211] F. Risso and I. Cerrato, "Customizing data-plane processing in edge routers," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 114–120.

[212] N. Kim, J.-Y. Yoo, N. L. Kim, and J. Kim, "A programmable networking switch node with in-network processing support," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2012, pp. 6611–6615.

[213] H. Gill, D. Lin, L. Sarna, R. Mead, K. C. Lee, and B. T. Loo, "SP4: Scalable programmable packet processing platform," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 75–76, Aug. 2012.

[214] M. Monti, M. Sifalakis, C. F. Tschudin, and M. Luise. (2016). "Towards programmable network dynamics: A chemistry-inspired abstraction for hardware design." [Online]. Available: https://arxiv.org/abs/1601.05356

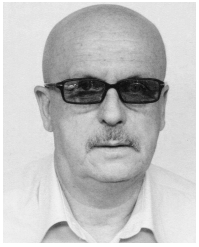
[215] M. Monti, M. Sifalakis, C. F. Tschudin, and M. Luise, "On hardware programmable network dynamics with a chemistry-inspired abstraction," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2054–2067, Aug. 2017.

[216] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: A retrospective on evolving SDN," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, New York, NY, USA, Aug. 2012, pp. 85–90.
[217] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *Proc. 11th ACM Workshop Hot Topics Netw.*, New York, NY, USA, Oct. 2012, pp. 7–12.
[218] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, "FlowTags: Enforcing network-wide policies in the presence of dynamic middlebox actions," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, New York, NY, USA, Aug. 2013, pp. 19–24.
[219] H. Mekky, F. Hao, S. Mukherjee, Z.-L. Zhang, and T. V. Lakshman, "Application-aware data plane processing in SDN," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, New York, NY, USA, Aug. 2014, pp. 13–18.
[220] B. Belter *et al.*, "Programmable abstraction of datapath," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, Sep. 2014, pp. 7–12.
[221] E. Haleplidis, J. H. Salim, S. Denazis, and O. Koufopavlou, "Towards a network abstraction model for SDN," *J. Netw. Syst. Manage.*, vol. 23, no. 2, pp. 309–327, Apr. 2015.
[222] S. Geissler, S. Herrnleben, R. Bauer, S. Gebert, T. Zinner, and M. Jarschel, "Tablevisor 2.0: Towards full-featured, scalable and hardware-independent multi table processing," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Jul. 2017, pp. 1–8.

ENIO KALJIC received the bachelor's and master's degrees in telecommunication engineering from the Faculty of Electrical Engineering, University of Sarajevo, Bosnia and Herzegovina, in 2008 and 2010, respectively, where he is currently pursuing the Ph.D. degree. From 2010 to 2014, he was a Teaching Assistant with the Department of Telecommunications, Faculty of Electrical Engineering, University of Sarajevo, where he has been a Senior Teaching Assistant, since 2014. His research interests include software-defined networking, and packet switching architectures for wireless and wired networking.

ALMIR MARIC received the bachelor's and master's degrees in electrical engineering from the University of Sarajevo, Bosnia and Herzegovina, in 2010 and 2013, respectively, where he is currently pursuing the Ph.D. degree with the Department of Telecommunications, Faculty of Electrical Engineering, and is also a Teaching Assistant. His current research interests include physical channel modeling, communication channel characterization, and fading channel and network simulators.

PAMELA NJEMCEVIC graduated from the Faculty of Electrical Engineering, University of Sarajevo, in 2006. She received the M.Sc. and Ph.D. degrees from the Faculty of Electrical Engineering, University of Sarajevo, in 2011 and 2016, respectively. Since 2006, she has been a Teaching Assistant and an Assistant Professor with the Faculty of Electrical Engineering and the Faculty of Transport, Traffic and Communications, University of Sarajevo. She was involved in the implementation of many research projects, related mostly to wireless communications and wireless propagation mechanisms. For two years, she was with EUPM BH, as a Junior EU PPU CARDS Expert for TETRA systems.



MESUD HADZIALIC received the Dipl.Ing., M.S.E.E., and Ph.D. degrees from the Faculty of Electrical Engineering, University of Sarajevo, Sarajevo, in 1978, 1986, and 2001, respectively. From 2009 to 2013, he was the Vice Dean of science and research and the Head of the Department of Telecommunications, Faculty of Electrical Engineering, University of Sarajevo, where he was a Full Professor with the Department of Telecommunications, from 2015 to 2018. He has authored or coauthored five textbooks of which three books university textbooks, several papers in journals, and over 50 papers at conferences. He led and participated in five scientific projects in the field of telecommunications supported by the Federal Ministry of Science of Federation of Bosnia and Herzegovina and over ten local and international projects in the domain of network simulations. He was the BiH Project Leader in projects co-funded by the European Union (South East Europe Transnational Cooperation Program). His research and teaching interests were in the general area of telecommunication techniques, theory and practice in the information networks, simulation methods, and techniques in telecommunication channels and networks.