A Survey on In-Network Computing: Programmable Data Plane and Technology Specific Applications

Somayeh Kianpisheh and Tarik Taleb

***Abstract*—In comparison with cloud computing, edge computing offers processing at locations closer to end devices and reduces the user experienced latency. The new recent paradigm of in-network computing employs programmable network elements to compute on the path and prior to traffic reaching the edge or cloud servers. It advances common edge/cloud server based computing through proposing line rate processing capabilities at closer locations to the end devices. This paper discusses use cases, enabler technologies and protocols for in-network computing. According to our study, considering programmable data plane as an enabler technology, potential in-network computing applications are in-network analytics, in-network caching, in-network security, and in-network coordination. There are also technology specific applications of in-network computing in the scopes of cloud computing, edge computing, 5G/6G, and NFV. In this survey, the state of the art, in the framework of the proposed categorization, is reviewed. Furthermore, comparisons are provided in terms of a set of proposed criteria which assess the methods from the aspects of methodology, main results, as well as application-specific criteria. Finally, we discuss lessons learned and highlight some potential research directions.**

***Index Terms*—In-network computing, programmable data plane, software defined networking, cloud computing, edge computing, 6G, and network function virtualization.**

I. Introduction

OVER the years, computation history has experienced evolution of various paradigms from traditional parallel, and grid computing to cloud computing. Cloud computing [1] that offers various service models including Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-as-a-Service (SaaS), brings advantages and capabilities such as scalability, on-demand resource provisioning, pay-as-you-go pricing model, and facilitated applications and services provisioning.

The era of 5G and beyond introduces a variety of new applications like mobile video conferencing, connected vehicles, e-healthcare, online gaming, and virtual reality. Blending the different research initiatives by industries and academia, these new applications demand high data rate in the scale of $1 \sim 100$ *Gbps* and low latency in the scale of $0.1 \sim 1$ *ms* round trip latency for ultra-low latency applications [2], [3]. Cloud computing can not sustain this ongoing requirements because of some issues. The main issue is that the distance between the cloud resources and the end device is large and the connection is established over the Internet which suffers from the aspect of latency. Furthermore, the processing capacity of cloud servers is in a range that can not compete with the emerging requirements. For example, the latest generation of general purpose computing instances in Amazon EC2 cloud service has processing capability in order of $5 \sim 50$ *Gbps* [4]. However, this processing capability can not efficiently be responsive for massive number of applications and the pattern of Internet of Things which generate huge traffic volume, competing in resource utilization for processing, where applications require high data rate (e.g., multiple-Gbps for high quality 360 degree video).

The idea of edge computing [5] with various paradigms of cloudlet, mobile edge computing and fog computing, was introduced to tackle the cloud related issues. Edge computing provides resources at the edge of network and closer to end devices. Though the latency will be improved and the processing capacity will be enhanced, it is unlikely to sustain the ongoing traffic explosion in the long run. Furthermore, the latency is still far from the required one for ultra-low-latency applications within less than 1 ms latency requirement, e.g., 0.1 ms round-trip latency.

The recent idea of distributed cloud computing improves the latency of cloudlet, mobile edge computing and fog computing paradigms by exploiting the computation and storage capacity of intelligent neighbourhood devices for computation or caching offloading [6], [7]. However, the computation and power limitation issues, mobility of neighbourhood devices, and more importantly the security aspects in computation offloading to neighbourhood devices are major challenges. A more secure, power-efficient, and stable computation fabric with high processing capacity can drastically improve the computation and be regarded as a complementary of the existent computational paradigms. In this direction, *in-network computing* paradigm, based on the programmable data plane technology (the evolved concept of SDN), can offer power-efficient with high processing capacity network elements at the edge of the network.

To simplify traffic engineering and network management, and to allow a more convenient development of new protocols

Manuscript received 24 May 2022; revised 21 August 2022; accepted 7 September 2022. Date of publication 14 October 2022; date of current version 24 February 2023. This work was supported in part by the European Unions Horizon 2020 Research and Innovation Program through the Charity and aerOS Projects under Grant 101016509 and Grant 101069732; in part by the Academy of Finland 6Genesis Project under Grant 318927; and in part by the Academy of Finland IDEA-MILL Project under Grant 352428. *(Corresponding author: Tarik Taleb.)*

The authors are with the Centre for Wireless Communications, University of Oulu, 019098 Oulu, Finland (e-mail: somayeh.kianpisheh@oulu.fi; tarik.taleb@oulu.fi).

Digital Object Identifier 10.1109/COMST.2022.3213237

and applications, the concept of Software Defined Networking (SDN) [8] was introduced in which the forwarding devices are decoupled from the control plane. According to SDN, the network intelligence and routing policies are applied through logically centralized controller in a software-based manner. Thus, the network elements that form the data plane are simple packet forwarding devices which are programmable through an open interface, e.g., OpenFlow [9]. SDN became enabler for an emerging technology of programmable data plane (PDP) [10], [11]. The fundamental feature of PDP is the capability of programming packet processing by means of some high-level languages. Therefore, unlike traditional way whereby fixed functions are bounded within the switch chip, PDP provides flexibility for network operators to have control over packet processing tasks as they would prefer; thus, ending to faster adoption of new data plane functions and facilitating development of prototyping. Furthermore, implementing new data plane functions, without re-designing the Application-Specific Integrated Circuits (ASIC) of switches, saves a significant of capital expenditure.

Network elements like switches and routers provide the connectivity between end-device and the edge infrastructure, as well as connectivity between edge and cloud infrastructure. Exploiting programmable network elements, not only for the purpose of connectivity, but also for the purpose of computation, is a new trend of computing paradigm namely called as *in-network computing* [12], [13]. Nowadays, programmable switches can process in the scale of billion packets per second at line-rate processing (e.g., 12.8 Tb/s processing capability in Intel programmable switch [14]), while supporting sub-microsecond packet processing delays [15]. Leveraging in-network computing, the packets are processed at line-rate, on the path and before reaching the edge/cloud servers. Indeed, in-network computing paradigm can offer faster processing facilities at locations closer to end devices, in comparison with edge or cloud servers employed by common edge and cloud computing paradigms. This paper provides a comprehensive survey on in-network computing. The contribution of this paper is as below:

(i) Providing the first survey on the subject of in-network computing.
(ii) Discussing the enabler technologies and protocols as well as discussing the hardware aspects.
(iii) Proposing a new categorization of studies based on the involvement of in-network computing in various applications and specialized topics. The proposed categorization also includes the applications in the scope of recent technologies, e.g., 5G/6G, edge computing, NFV.
(iv) Critically reviewing the related studies and proposing novel methodology/performance/application-related criteria for the purpose of evaluating and comparing the studies.
(v) Perusing and comparison of the related studies from the aspects of methodology, implementation, and performance gains due to in-network computing, i.e., latency/throughput enhancement, bandwidth saving, power consumption reduction.
(vi) Providing lessons learned and research directions for the infancy topic of in-network computing.

In the following sections, we first give a definition of the in-network computing. Then, we discuss existing relevant surveys and tutorials. Following that, literature classification and survey organization will be given. In the rest of survey, we use INC for abbreviation of In-Network Computing.

A. In-Network Computing Definition

In this section, we first provide a brief introduction of network elements involved in the concept of in-network computing, then we provide definitions of in-network computing in the literature, finally, we will introduce some features to specify in-network computing and will give a schematic of computing capabilities provided by in-network computing. In-network computing paradigm advocates the idea of exploiting network elements, i.e., programmable switches, FPGAs, and smart NICs, to be programmed for the purpose of computation. Programmable switches have the capability of being programmed to parse and manipulate the arrived packet fields. FPGAs are semiconductor elements with the capability of programming logical blocks to perform targeted processing on the packets. Similarly smart NICs offer the implementing of dedicated hardware acceleration functions as well as customized packet processing. Section II-A gives details about architecture and functionality of these network elements.

There is no standard definition for the in-network computing. A definition by ACMSIGARCH is provided in [16]: In-network computing refers to the execution of programs typically running on end hosts within network elements. It focuses on computing in the network, using devices that already exist within the network and are already used to forward the traffic. Sapio et al. [12] gives a definition based on offloading computation to the network elements: In-network computing is offloading a set of compute operations from end hosts into network elements such as switches and smart NICs. Ports and Nelson [13] define in-network computing as: application-specific functions that can run in programmable network hardware at line rate, offering orders of magnitude higher throughput and lower latency than can be achieved by a traditional server.

According to the definitions and our surveyed studies, some features can be defined for in-network computing: (i) It focuses on computation that is performed in network elements. This computation can be a program, a process, operations, network functions; (ii) The involved network elements (e.g., switches and routers, FPGAs, and smart NICs) can be programmed to perform the expected computation; (iii) Beside the computation, the network elements perform routing and forwarding packets as their default procedure; (iv) In the case of not using the network, the computation supposed to be done by general purpose processor either in an application host or any other end host (e.g., server, controller).

According to our surveyed studies, we propose Fig. 1 to illustrate the schematic of computing capabilities provided by in-network computing. The in-network computing fabric consists of network elements in the blue cloud, which can

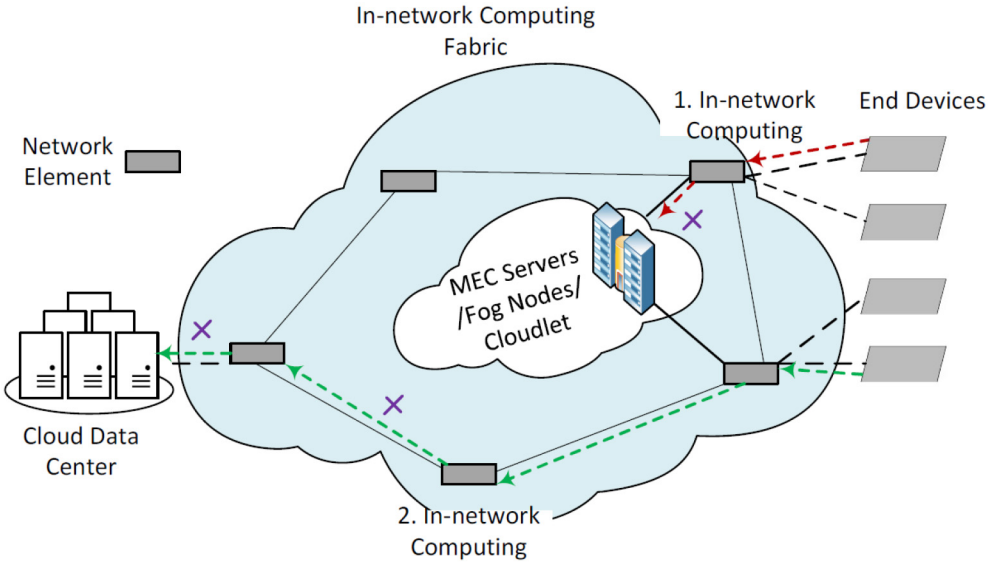

Fig. 1. Schematic of in-network computing fabric. Network elements in blue cloud, form in-network computing fabric. The red and green paths respectively illustrate communications with edge and cloud. As showed by cross sign, these paths can be terminated at points 1 and 2, prior to edge/cloud servers, by applying in-network computing in the middle network elements.

be located among end-devices and servers provided by edge computing (e.g., MEC servers, fog nodes, cloudlets), as well as among end-devices and cloud data center (or even among edge server and cloud server). The green path illustrates an end-to-end communication path which can be truncated at a point prior to servers at cloud (e.g., point 2), by applying in-network computing in the middle nodes on the path at network elements, which can be programmable switches, or FPGAs, and smart NICs accelerators embedded in a host. Similarly, the traffic following the red path might be processed at the network element prior to edge servers (e.g., point 1) and the results be returned to the end-device from a closer distance than edge server.

B. Existing Relevant Surveys and Tutorials

In this section, we discuss the existent relevant surveys and tutorials, and discuss the contribution of our survey in comparison with them.

1) Surveys/Tutorials on Programmable Data Plane: The surveys/tutorials in this category focus on programmable data plane and lack the view of in-network computing. In comparison with these surveys, there are four main differences that clearly distinguishes our survey from the existing studies: first, we propose a novel categorization in the scope of in-network computing based on the application and specialized topics the in-network computing is involved; second, we provide a comprehensive and critical review of in-network computing papers including the applications in the scope of recent technologies, e.g., 5G/6G, edge computing, NFV; third, we introduce technologies, protocols, and hardware aspects that enable in-network computing; fourth, we provide evaluations and wide-comparisons of our surveyed studies from the aspect of our proposed methodology/performance/application-related criteria that have not been seen in the literature. There are also other differences that we will highlight as we discuss the surveys/tutorials on programmable data plane.

Stubbe [17] provides a short survey on P4 compiler and interpreter. Bifulco and Rétvári [10] provides a survey on the abstractions, architectures and issues in the design and implementation of programmable network elements. Han et al. [11] give an overview of existing PDP virtualization schemes and discuss their pros and cons. Kaljic et al. [18] provides a survey on data plane programmability and flexibility in software defined networking. Data plane architectures are evaluated through data plane flexibility and programmability aspects. Based on assessing the limitations of ForCES and OpenFlow data plane architectures, some approaches to address the data plane flexibility and programmability issues are given. However, the articles in [10], [11], [17], [18] lack a review of the existing applications over programmable data plane, the related challenges, and potential research directions.

The articles in [19], [20] focus on stateful data plane. Zhang et al. [19] provides an overview of basic components of stateful data plane, and existing stateful platforms (e.g., OpenState, OPP, FAST, etc.). The article reviews a few applications based on stateful data plane, e.g., load balancing, firewall, SYN-flood detection, heavy-flow detection. Dargahi et al. [20] provides an overview on stateful SDN data plane studies and focuses on the security aspects of data plane programmability. The authors identify some attack scenarios, and highlight some vulnerabilities for stateful in-switch processing. The survey, however, does not discuss existing applications other than security, the related challenges, and potential research directions.

Da Costa Cordeiro et al. [21] describe prominent programming languages that enable data plane programmability. The authors consider two categories of data plane programmability literature: 1) programmable security and dependability management, including studies for policy modeling and

analysis, policy verification, intrusion detection and prevention; 2) enhanced accounting and performance management, including studies on network monitoring, traffic engineering, and load balancing. The survey only discusses a limited number of papers.

Kfoury et al. [22] provide an overview of the network evolution from legacy to programmable, describe the role of programmable switches and P4, and review applications developed with P4, e.g., network telemetry, Internet of Things, network performance, network and P4 testing. Although this article gives more details than all the other tutorials/surveys in the category of programmable data plane, there is still some major differences between our survey and this study. Our survey proposes a different taxonomy which is categorized based on in-network computing application as well as specialized involved topics. Furthermore, there are considerable amount of in-network computing papers that have not been reviewed by [22]: For example, papers in the scopes of recent technologies, e.g., 5G/6G, NFV and edge computing (Please see Section VII of our survey for the wide range of papers we covered), also papers in other scopes such as [23], [24], [25], [26], [27], [28], [29] etc. In addition, we have evaluated and compared the studies from the aspects of novel methodology/performance/application-related criteria which have not been considered in [22]. Finally, the research directions in [22] mostly focus on the data plane solutions to overcome the constraints of programmable switches, e.g., switch resources, arithmetic computation, programming. In contrast, we provide research directions for various applications, which give insights on the research gap in various categories of applications and related research directions.

2) Tutorials on In-Network Computing: To the best of our knowledge, there is no survey for in-network computing and our study provides the first survey. There are three short tutorials, i.e., [13], [30], [31], that list a few number of papers in the scope of in-network computing without providing explanations regarding the operation and main ideas of each paper or giving any taxonomy for the studies, technical details, comparison, lessons learned, or a concrete research direction. Our survey can be easily distinguished from these tutorials since the existent tutorials are in the scale of a few pages. Hereunder, we give more details on these tutorials.

Ports and Nelson [13] present a short tutorial on in-network computing. It introduces programmable network elements and discusses effective usage of in-network computing. Then, it classifies a limited number of in-network computing applications and suggests appropriate network elements to implement them. Benson [30] present a brief overview of management challenges for in-network computing and explain the limitations of existing management techniques. It only gives a short list of in-network computing applications without any review. Kannan and Chan [31] present a short tutorial on the evolution of programmable networks starting from efforts before software defined networking to the more recent programmable data plane. They discuss the advantages of data-plane programmability. The authors categorize data plane applications into two categories, i.e., network monitoring and in-network computing. This tutorial only lists a number of papers belonging to the aforementioned categories and does not provide any review of them. Also, there is no structure for the listed papers in the category of in-network computing. The reader can easily surmise, none of the aforementioned tutorials offer review or categorization of the in-network computing papers in the literature, or offer any discussion about enabling technologies and protocols for in-network computing. This comprehensive survey aims to cover this gap.

C. Literature Classification and Survey Organization

This manuscript provides a survey of the existing papers (i.e., in the context of algorithms, protocols, and architectures), presented in peer-reviewed venues, in the scope of in-network computing. The literature is reviewed using a set of well-defined criteria. In this section, first the literature classification is discussed, and we then describe how the survey is being organized.

1) Literature Classification: We have identified a total of one hundred-six papers to be reviewed in this survey. We have collected the papers that meet any of these criteria: (i) Explicit usage of the term in-network computing in the title, (ii) Explicit usage of in-network computing as defined in Section I-A, as the motivation of the paper. Fig. 2 illustrates the proposed taxonomy of the survey. As the papers in the scope of in-network computing have provided in-network solutions for various applications, at the first level of taxonomy we have categorized the papers based on their applications we found in the literature, i.e., (i) in-network analytics, (ii) in-network caching, (iii) in-network security, (iv) in-network coordination, and (v) technology specific applications. Then, at each category, we have categorized papers according to the specialized topics that in-network computing will be involved within the context of that category.

The motivation behind this categorization is that this categorization clearly defines the involvement of in-network computing. Indeed, considering each leaf in Fig. 2, the reader can recognize the hierarchy of specialized topics within a specific application, at which in-network computing will be involved. Here, we provide three examples: First, considering, *Flooding* in Fig. 2, it would be clear that in-network computing has been involved to provide solution for the purpose of flooding attack mitigation, which is in the scope of DDoS attack mitigation, and generally is for the application of security. Second, considering *Edge Intelligence*, it would be clear that in-network computing has been involved to provide edge intelligence, which is in the scope of edge computing, and generally is for a technology specific application. Third, considering *Cloud Empowered With INC: Resource Allocation Studies*, it would be clear that in-network computing has been involved in empowering cloud computing when resources are allocated, which is in the scope of cloud computing, and generally is for a technology specific application. Similarly, For each leaf in Fig. 2, the involvement of in-network computing in specialized topics and application can be found by tracking the taxonomy from leaf to the root. Now, we provide more details on the structure of each category and give some

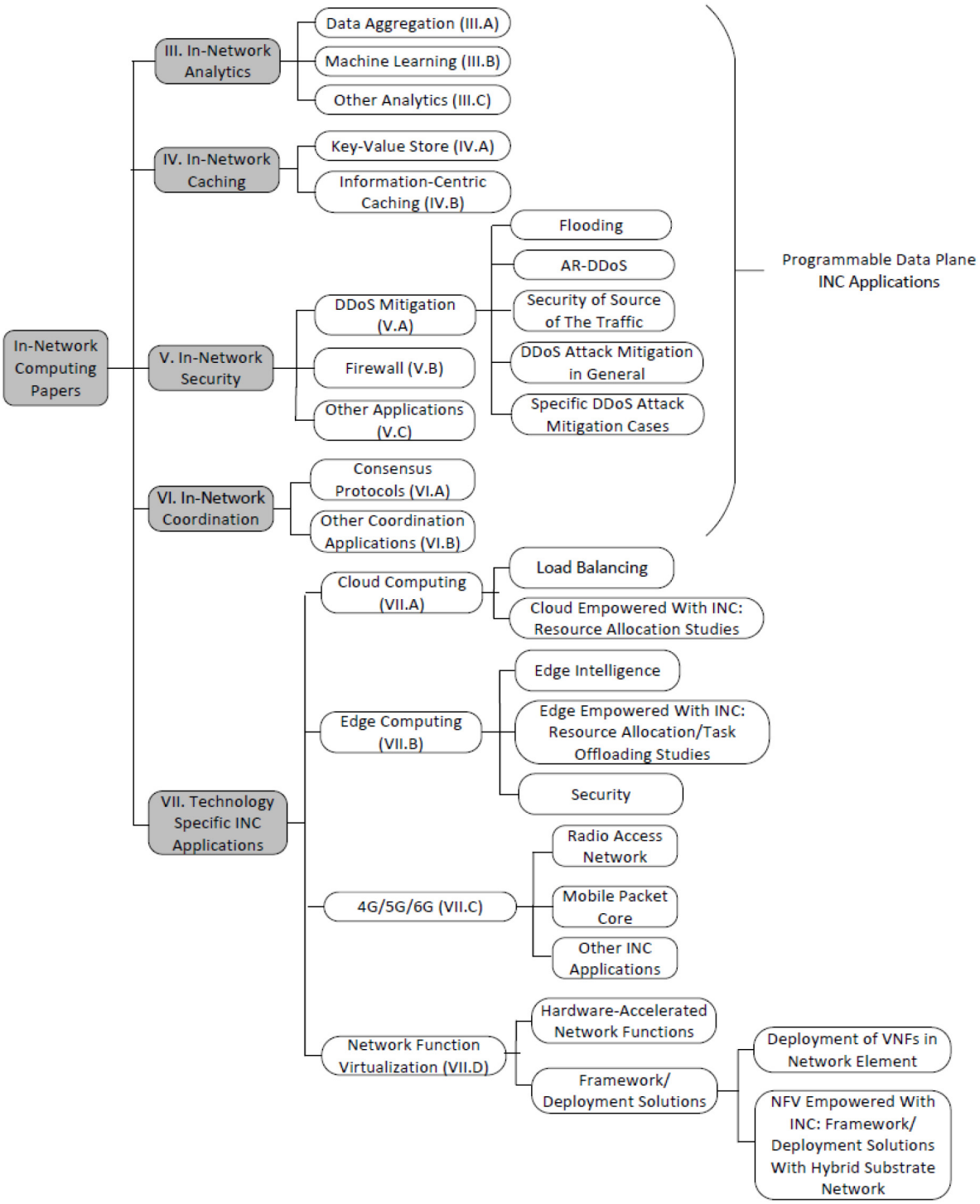

Fig. 2. The proposed classification for the surveyed studies in the scope of in-network computing. The numbers within boxes indicate the sections and subsections.

complementary explanations for the semantic/reason behind division of papers within categories. Please note that as the in-network computing topic is in infancy state, we could extend the specialized topics at each category up to the point that we found papers in that category.

The first category, i.e., in-network analytics based on the type of analytics is structured to three subcategories: (i) data aggregation, (ii) machine learning, and (iii) other analytics.

In the second category, i.e., in-network caching, we found that the papers provide caching based on key-value store. Furthermore, the topic of information-centric caching is also relevant to in-network caching. Thus, we have identified two subcategories: (i) key-value store, and (ii) information-centric caching.

In the third category, i.e., in-network security, most of the papers exploit in-network computing for DDoS attack

mitigation. There are also studies providing firewall solutions. There are also papers that provide a variety of security applications. Thus, in-network security is structured to three subcategories: (i) DDoS attack mitigation methods, (ii) firewall methods, and (iii) other security applications. In DDoS attack mitigation, based on the attack type we recognized five subcategories, i.e., (i) methods for flooding attack mitigation, (ii) methods for AR-DDoS attack mitigation, (iii) methods that consider security of the source of the traffic, (iv) methods that consider DDoS attack mitigation in general, and (v) specific DDoS attack mitigation cases.

The fourth category, i.e., in-network coordination comprises two subcategories: (i) consensus protocols, and (ii) other coordination applications.

In the fifth category, i.e., technology specific INC applications, based on the technology, we identified four subcategories: (i) cloud computing, (ii) edge computing (iii) 4G/5G/6G, and (iv) network function virtualization.

In cloud computing, we identified two subcategories: (i) papers that provide load balancing for data centers. (ii) papers that give resource allocation solutions in cloud computing empowered with INC.

In edge computing, we identified three subcategories: (i) papers that provide edge intelligence solutions. (ii) papers that give resource allocation solutions in edge computing empowered with INC. (iii) papers that provide security solutions for edge computing.

As radio and core division in the scope of 4G/5G/6G is a well-known division in the literature of mobile communication, we also found it appropriate for introducing in-network computing involvement in radio access and core functionalities. Thus, 4G/5G/6G category is comprised of three subcategories: (i) radio access network solutions. (ii) mobile packet core solutions. (iii) other applications.

In the scope of NFV, papers provide framework or deployment solutions in NFV. Furthermore, there is a topic of hardware-accelerated network functions which is in the scope of NFV and relevant to in-network computing. Thus, we identified two categories: (i) hardware-accelerated network functions. (ii) framework/deployment solutions which is structured to two sub categories: (i) the papers that focus on deploying VNFs in network elements, (ii) the papers that provide deployment solutions with a hybrid substrate network in a NFV environment empowered with INC.

2) Survey Organization: Section II discusses enabling technologies and protocols for in-network computing. Then, we present two use cases (i.e., in-network analytics, and in-network caching) to illustrate the concept of in-network computing, followed with in-network computing benefits. Finally, we propose our criteria to categorize, evaluate, and compare in-network computing papers in the literature. In Section III, we provide an overview of research on in-network analytics including data aggregation methods, machine learning methods, and other analytics in the network. An overview of in-network caching methods, composed of key-value store applications, and information-centric caching will be given in Section IV. In Section V, we provide a review of the literature in the scope of in-network security including DDoS attack mitigation methods, Firewall methods, and other security applications. Section VI gives a review on in-network coordination methods including consensus protocols and other coordination applications. In Section VII, we review technology specific applications including cloud computing, edge computing, 4G/5G/6G, and network function virtualization. Fig. 2 illustrates the organization of the survey at Section III till Section VII. Furthermore, at the end of each section, we provide summary, comparisons and lessons learned for that section. In Section VIII, we discuss some potential research directions. Finally, the paper concludes in Section IX.

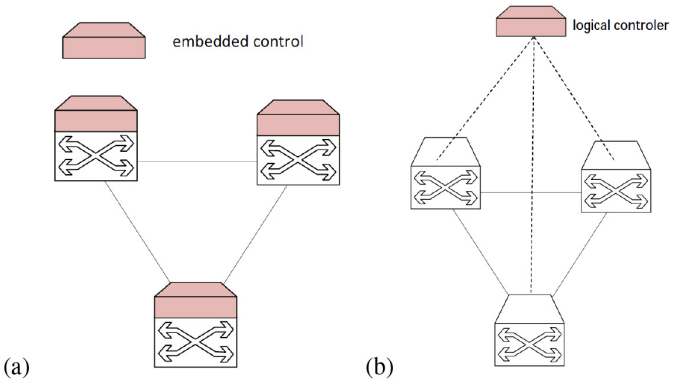

Fig. 3. (a) Traditional network, (b) Software defined network.

II. Technologies, Protocols, Illustrative Use Cases, and Criteria

In this section, we first explain the technologies and message forwarding protocols that enable the in-network computing. We then provide two use cases to illustrate the concept of in-network computing, followed with some highlights on in-network computing benefits. Finally, we explain the criteria according to which we analyze and compare the studies in the survey.

A. Enabling Technologies

1) Software Defined Networking: Software Defined Networking (SDN) is a networking paradigm emerged to facilitate remote network management as well as deployment of new routing policies, protocols and applications. According to the concept of SDN, the forwarding hardware in the data plane is decoupled from control decisions. Fig. 3a and Fig. 3b show the concept of SDN in comparison with embedded controls in devices in the traditional networks. The intelligence for network operation and routing policy is logically centralized through software-based solutions developed in the control plane, and network elements become simple packet forwarding devices forming the data plane, that can be programmed via open interfaces (e.g., OpenFlow [9], ForCES [32], etc.)

OpenFlow [9] is the most popular SDN protocol. In this protocol, the forwarding device, i.e., OpenFlow switches, contains several flow tables as well as an abstraction layer that communicates with a controller through OpenFlow protocol.

Each flow table contains several flow entries for the purpose of defining packet processing and forwarding strategies. Generally, flow entries consist of: a) match fields/rules containing information found in the packet header, ingress port, and metadata, which are used to match incoming packets; b) counters, used for the purpose of collecting statistics of the flow (e.g., number of arrived packets, flow duration); and c) a set of actions, for the purpose of being operated on the packets when a matching occurs. When a packet arrives at an OpenFlow switch, the matching of packet header fields with the matching fields is investigated. If a matching occurs, the associated set of actions are applied. Flow entries can be added, updated, or deleted from the switch's flow tables, by the controller upon its communication with the switch via OpenFlow protocol. Focusing on the controlling aspect, scalability and performance of the network controller, reactive or proactive communication between SDN controller and switches, and the implementation of southbound and northbound communication with respect to the forwarding elements and network services define some of the challenges in designing an SDN controller. ONOS, OpenDaylight, Floodnight [33], Beacon [34], and RouteFlow [35] are examples of SDN controller implementations.

2) Programmable Data Plane: The main elements involved in the technology of programmable data plane are as below:

(i) Separation of Control Plane From Data Plane: A main contributor for data plane programmability was introduced by the concept of the decoupling of the control plane from the data plane, with a standard API for the purpose of interactions between the two planes [8], [36]. Indeed, the data plane can be realized by "dumb" network element under the administration of a control plane. However, these "dumb" network element expose embedded state information to the control plane which enables network programmability.

(ii) Data Plane: The data plane is the most fundamental infrastructure of a network that processes the packets received or delivered by the network elements. A combination of hardware components and specific software methods are utilized to implement the data plane. The data plane functionalities may be implemented within various network elements, e.g., ASIC, FPGA, network processor, NIC, based on a packet classification engine. Network element technologies expose the packet processing primitives to the control plane in various ways and use various programming languages for accessing packet processing primitives. Here, we discuss more details about the hardware and functionality of network elements that are involved in in-network computing.

(a) Programmable Switch: SDN switches are categorized into hardware and software switches. Barefoot Tofino [37], Cavium XPliant [38], and Flexpipe [39] are examples of hardware switches. On the other hand, a software switch performs the packet processing logic on a CPU based on a packet classification algorithm [10]. OVS [40], PISCES [41], NetBricks [42], and BMv2 [43] are examples of software switches. The functionality of a programmable network switch, can be abstracted as a match-action pipeline. Protocol Independent Switch Architecture (PISA) is a renowned match-action based architecture for programmable switches [11]. Fig. 4a shows PISA.

The architecture is composed of programmable substructures including parser, match-action pipeline, and deparser. The parser extracts headers from the arrived packets, stores them in intermediate registers called Packet Header Vectors (PHVs) [44], which are regarded as input to a pipeline of stages. At each stage of pipeline, the extracted headers are processed using match-action tables. In hardware switches match-action tables are implemented in Ternary Content Addressable Memory chips (TCAM), while in software switches they are implemented in SDRAM. A hybrid case at which match-action tables be located at both TCAM and SDRAM is also possible. When matching does not occur, the packet will be sent back to the device/SDN controller, and accordingly the required updates in the table entries will be performed. When there is a matching, the action will be performed and the packet will be sent to the next match-action table for further required processing. At each match-action table, the action logic is applied by Arithmetic Logic Units (ALUs). Through an action, an operation will be performed on the packet fields and the result will be stored in PHVs. Other objects like counters, or registers that are stored in the SRAM can be used to perform stateful actions which operate on the output of previous actions. After the completion of process at the pipeline stages, the processed headers are sent to the deparser, that will combine the headers to reconstruct the packet. A control plane can manage packet processing by writing entries in the match-action tables.

PISA provides an abstract model that can be boosted in various ways to create a concrete architecture. For example, a typical switch can have separate pipelines for ingress and egress, and a Queues manager module, i.e., scheduler located between ingress and egress pipelines (Fig. 4b). Furthermore, specialized components for advanced processing, e.g., hash/checksum calculations can be also included in the concrete architecture. The state of the art programmable switches can process in the scale of billion packets per second at line-rate processing [15]. As an example, Intel second-generation P4 programmable Ethernet switch ASIC can deliver up to 12.8 Tb/s throughput [14]

(b) FPGA: FPGAs are semiconductor elements with the capability of programming and configuring after manufacturing to implement required packet processing. As illustrated in Fig. 5 the hardware architecture of a FPGA consists of three main elements [45]: (i) Compute Logic Blocks (CLBs), (ii) routing capabilities (illustrated as solid lines connecting CLBs), (iii) I/O blocks. Through look-up tables and flip-flops, CLBs provide a programmable matrix of compute units terminating at the I/O blocks. The I/O blocks can be connected to interconnects like PCIe through which the communication with CPU and system is provided. The routing capability which is realized through interconnect components like switch boxes and connection boxes, provides connectivity among CLBs to create facilities for generating a complex logic of computation. In comparison with specific ASIC designs which has the highest performance, state-of-the-art FPGAs which exploit high clock speeds and memory bandwidth, has shown to narrow this performance gap for many use

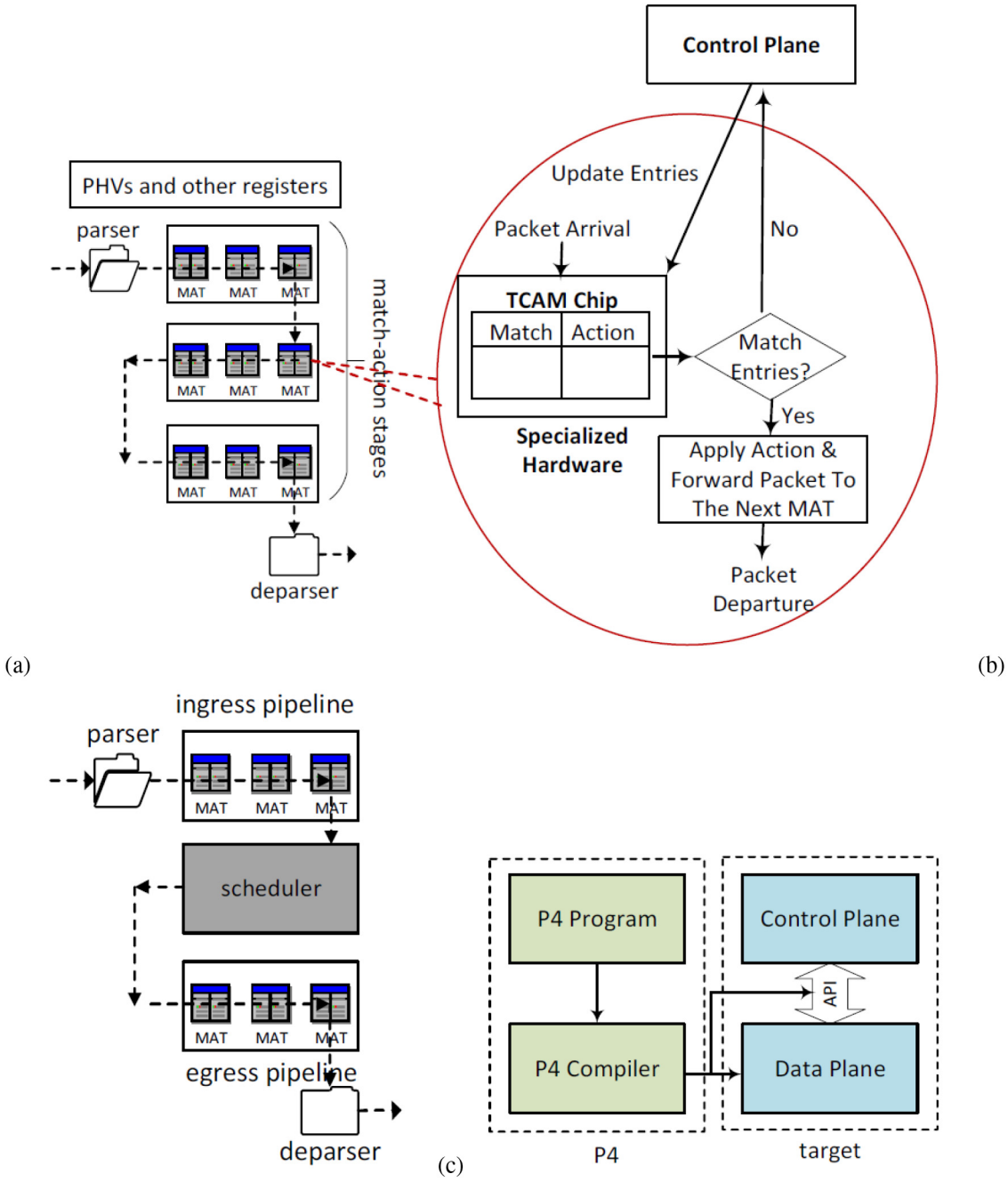

Fig. 4. (a) Protocol independent switch architecture, (b) A typical switch architecture, (c) Relation of P4 program with target.

cases while having the advantages of flexibility and cost-efficiency [46]. A well-known state-of-the-art version, FPGA SUME exploits a Xilinx Virtex 7 FPGA with four 10 Gb Ethernet ports [47]. A more recent FPGA-based prototyping platform is Corundum [48], with capability of provisioning a 100 Gbps NIC on FPGA.

(c) Smart NIC: The NICs are external hardware components that can be connected to a computing node through the PCIe interfaces. Some standard physical and MAC layer functionalities, as well as some Internet protocol layer functionalities are implemented in NICs. Indeed, receiving and transmitting packets from/to the Internet, as well as initial processing of IP packets prior to delivery to operating system or application layer are performed by NIC. The NIC technology is advanced with smart programmable NICs, which enables programming of NICs with general-purpose or specialized data plane languages (e.g., P4, eBPF). Not only dedicated hardware acceleration functions are implementable in NIC, but also general-purpose packet processing, like FPGA units can be programmed to perform customized processing. In comparison with other accelerators (e.g., FPGA), direct processing of packets at NIC after their arrival, can omit the delay due to the transferring of packets from system memory to accelerator memory in order to be processed. However, the adoption of NIC for in-network computing still copes with challenges due to development process of applications as well

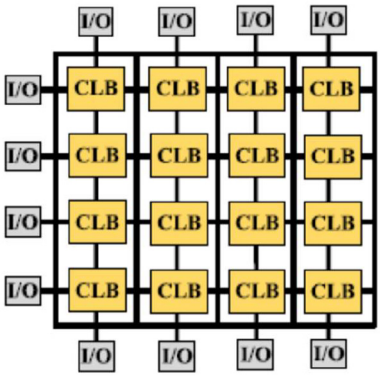

Fig. 5. FPGA architecture. [45].

as performance issues [46], [49]. In particular, to address the development abstraction in server application offloading, Floem [50] provides programming abstractions for the purpose of data placement, caching, and parallelism, and communication policies between program components across devices. The state-of-the-art SmartNICs offer high-speed packet processing on the order of 400 Gb/s [51].

(iii) Data Plane Programming Languages: There exists various data plane programming languages in the literature such as P4 [52], OpenState [53], Domino [54] and NetKAT [55] among which, P4 is the most widely used programming language. P4 is a high-level programming language that is used for processing packets in programmable network elements [52], [56]. While P4 was initially designed for programmable hardware or software switches, its scope has been extended to support a large variety of other devices including NICs, and FPGAs. Thus, in the specification of P4, the generic term *target* is used for all such devices. In comparison with traditional fixed function switch at which the functions are defined by the respective manufacturers, the switch functionality can be defined by a P4 program. The control plane will communicate with the data plane through the API that is generated by the P4 compiler to bring flexibility in the usage of tables and other objects in the data plane.

P4 has been designed to follow three main requirements [56]: (i) To enable the controller to reconfigure the packet parsing/processing in the switch, (ii) Specifying a packet parser/processing for a general context of packet forwarding and independent of the used protocol, (iii) To do the programming of the switch independent of the details of the underlying switch. A P4 program contains these main components: (i) *Header* to define the sequence and structure of fields as well as the constraints on the field values, (ii) *Parser* to specify headers and their sequences within packets, (iii) *Match-action Tables* to define packet processing, (iv) *Actions* that are applied on matched fields, and with the capability of making complex actions from simpler primitives, (v) *Control Programs* which determine the order and flow of control of match-action tables that are applied to a packet. P4 compiler maps the program description into the target's specific hardware/software platform. The compilation process consists of two-stages. First, the P4 control program is transformed into a table dependency graph structure that defines the dependencies among the tables. Then, through a target-specific map, the aforementioned graph is mapped onto the switch's specific resources. Fig. 4c illustrates the relation of P4 program with the target.

3) Edge Computing: edge computing (also referred to as cloudlet, mobile edge computing or fog computing) provides resources at the edge of networks, in the proximity of end-devices, in order to reduce latency and enable capabilities such as mobile data offloading [5]. In the rest of this section, we provide an overview of edge computing paradigms:

Cloudlet: cloudlet, proposed by Satyanarayanan et al. [57], are clusters of servers that are located close to mobile devices. Mobile devices can offload their computations to Virtual Machines (VMs) running in the cloudlet, in order to overcome the limited available resources at the devices. Being based on VM technology, resources in a cloudlet can expand and shrink dynamically, and will have scalability with respect to the service requests. Mobile devices can offload their computations to the cloudlets in their proximity, thereby overcoming the poorness of resource limitation in the device, as well as guaranteeing real-time interactive responses. If the cloudlet is not accessible in the proximity of the mobile device, there is still possibility of connecting to a distant cloud, however there will be a response time degradation in getting the required service.

Multi-Access Edge Computing (MEC): MEC, introduced by the European Telecommunication Standards Institute (ETSI), was initiated under the name of Mobile Edge Computing (MEC). Initially, it included mobile networks and VM as virtualization technology. However, later, the idea was expanded to support also non-mobile network requirements, as well as including other virtualization technologies. Indeed, MEC offers cloud computing facilities at the edge of the network through mobile edge computing servers which are accessible by LTE macro base stations (eNodeB) and multi-radio access technology sites.

Fog Computing: fog computing extends cloud computing from the core to the edge of the network, and thereby, providing the computing facility at the edge of the network, closer to the end-devices. Fog computing deploys fog nodes (e.g., edge switches, gateways, smartphones, access points, etc.,) close to the user and in a layer between the end user and cloud. Unlike cloudlet and MEC, fog does not operate in a standalone mode and is coupled to the existence of cloud.

Decentralized Cloud computing: The recent idea of decentralized cloud computing can be seen as an evolved version of edge computing that exploits the computation and storage capacity of intelligent devices (e.g., smart phone, cell phones, sensors, drones, cars, etc.) to enhance the computation/storage capacity of edge computing and provide the cloud capabilities in the neighbourhood of devices. The computing continuum introduced by this paradigm, not only enhances the latency experience, but also alleviating the problem of edge servers/data storage overloading, as well as high cost due to edge infrastructure deployment. However, the computational/storage and energy constraints, as well as mobility of devices, and security aspects in computation/content

offloading to neighbourhood devices are still the major challenges. For more details, interested reviewers are referred to surveys [6], [7].

4) Roles of Technologies in the Emerging In-Network Computing: In this section, we explain how the mentioned technologies play a role to enable in-network computing:

Programmable Data Plane: programmable data plane provides the infrastructure to perform computation inside the network. The required computation (e.g., data analytics, security policies, caching, load balancing, etc.) are implemented in programmable data plane. Programmable switches and routers, FPGAs, and smart NICs are examples of programmable data plane elements that have been used in the scope of in-network computing.

Edge Computing: Edge computing can contribute to in-network computing by providing in-network computing equipment, i.e., network elements at the edge in order to carry out the required computation on the path before reaching remote servers. Indeed, edge computing empowered with in-network computing has been discussed in several studies, e.g., [15], [58]. In this regard, the computation can be done on the path and closer to the end device in comparison with common edge servers.

Software Defined Networking: Performing computation, decision, or controlling at network elements as advocated in the concept of in-network computing, has the potential to reduce the SDN controller intervention as discussed in [59], [60]. However, SDN controller can still perform inevitable management tasks as well as the tasks which can be performed efficiently based on the global view of the system, e.g., installing and updating rules in data plane, configuring and installing programs and functions on data plane, configuring network to block malicious traffic, validation etc. [60], [61], [62], [63], [64].

B. Enabling Protocols

Up to date, and to the best knowledge of the authors, no specific communication protocol has been proposed in the literature to facilitate communications for a wide-range of in-network computing applications. Thus, the existent studies, in the scope of in-network computing, use the already existing message passing mechanisms. Message passing mechanisms which are not based on IP addressing for the delivery of messages are promising to realize in-network computing capabilities. In this section, we discuss four existent non-IP based message forwarding mechanisms and explain how they can enable in-network computing.

1) Uninformed Message Forwarding: Uninformed routing algorithms is a forwarding protocol mechanism that does not use the knowledge of query semantics or destination node's address in the forwarding decisions. In this direction, flooding [65] and random walk [66] are the most popular algorithms. These algorithms can be used to flood the message in the network so that network elements can examine the message and apply the required *computation* in the case that they can implement the *computation*. However, these methods are not efficient in terms of the traffic volume they produce.

2) Information-Centric Networking: The fact that the Internet is extensively used to disseminate information and data, rather than used for pair-wise communication between source and destination, became a principle for the idea of Information-Centric Networking (ICN) as a possible architecture for the future Internet. In this direction, the first idea was introduced by Gritter and Cheriton [67]. ICN advocates the deployment of in-network caching, as well as multicast transmission, to offer a more efficient delivery of information to the users. Based on ICN, the information is named and matched independently of its location, thereby it may be provided from anywhere in the network. Upon a request arrival, the network will locate the best source that can provide the desired information. Interested readers are referred to [68] for a detailed survey on ICN. In the same direction of ICN, a general view of information as a *required computation*, and naming as a *computation name*, makes it possible to query for *computation* and accordingly, the *computation* can be provided by the network elements despite of the location of the device.

3) Service-Centric Networking: Service-Centric Networking is an extended version of ICN, which provides supporting for both content and computation services for the future Internet, e.g., 6G [69]. Each computation service can be identified by a unique name to indicate function(s) and parameter(s). It uses a three-phase operation to execute a function: (i) forwarding the request toward the function, (ii) fetching the required data, and (iii) computing and returning the result. Furthermore, it supports chaining among functions to serve more complex services. This paradigm of computing, lets in-network computing be performed to execute functions without knowing locations.

A content/computation/context-aware adopted version of service-centric networking protocol has recently been presented in [69] to be promising for 6G. An Interest/Data name requesting comprising service identifier, target object, and context is used for forwarding and knowledge purposes. After the service identifier, the target object indicates the forwarding direction, while the Context object provides additional information for a function computation. When an Interest arrives, the content store of a data plane element is searched to obtain a cached content or computed result based on the context information. In the case of cache miss, the protocol lets also the local computing of the Interest in data plane element, i.e., in-network computing or forwarding the Interest through a forwarding pipeline procedure. The SDN controller is also adopted by supporting new match fields (i.e., service, object, and context), more advanced forwarding techniques to handle caching, executing functions, and function chaining, as well as design of new flow tables according to the context of Data/Interest processing.

In light of provisioning ultraLow-Latency (ULL) in new emerged 6G vertical services (such as URLLC applications, e.g., autonomous driving, industrial control), as suggested in [69], the ultra-low-latency requirement can be inserted as part of service request in Interest/Data name protocol, in order to apply corresponding in-network computing/caching, as well as latency-aware forwarding strategies. As latency-aware packet forwarding is out of scope of this paper, we call here

some standardization attempts. The IEEE 802.1 time sensitive networking standard provides link layer support for ULL networking, while IETF deterministic networking standards provides the complementary network layer ULL support. Interested readers are referred to [70] for a survey on the IEEE TSN and IETF DetNet standards and the related research studies.

3) SemanticBased Message Forwarding: In semantic based message forwarding, the messages are routed based on their meaning instead of IP addresses [71], [72]. To send a message to a particular network node, a semantic key with the meaning similar to the description of the targeted node in the network is inserted into the message. Once the message arrives to the network, it is delivered to the intended destination defined by the semantic key, and then the destination can respond back to the message source. Indeed, the network can be considered as a collection of interconnected semantic routers where each router compares similarity between semantic keys in the messages and resource description stored in semantic routing tables to decide about the next hop destination.

Another form of semantic routing is content routing which has been utilized as a routing mechanism in peer-2-peer networks [71]. Content routing algorithms also exploit the semantic information which is embedded in user query, for making routing decisions at each hop. In content routing, semantics or objects are identified by keywords, and advertisements and queries are expressed in terms of these keywords. In contrast with address based routing, in semantic routing, objects are identified by keys, which are constructed by applying hash function on the keywords associated with the objects. As a key based routing will select a specific resource having the key to handle the message, it is more efficient than keyword-based message routing. However, the key-based query routing does not support partial matching semantics. In contrast, content routing systems can support partial-matching queries through utilizing blind search methods. However, the generated query routing traffic would be high and there is no guarantee on search completeness.

One of the most commonly used content routing techniques is intelligent flooding. It forwards a message to some of the neighbors based on some criteria like previous query results, capacity of nodes, type of content, etc, thereby reducing the overhead of blind searching method. Ahmed et. al provide a detailed description of content routing [71]. In the context of in-network computing, semantics can be considered as *computations* which can be implemented by network elements. To facilitate search mechanism, network elements can expose their supported *computations* to a data structure similar to semantic routing tables. Based on the semantic, i.e., *computation* information within the message, such data structure can be utilized to route messages toward the network element that can apply the *computation*.

C. Illustrative Use Cases and In-Network Computing Benefits

In this section, we illustrate in-network computing concept through two use cases in the scope of in-network analytics and in-network caching. For each use case we discuss the procedure without in-network computing, and the procedure with in-network computing.

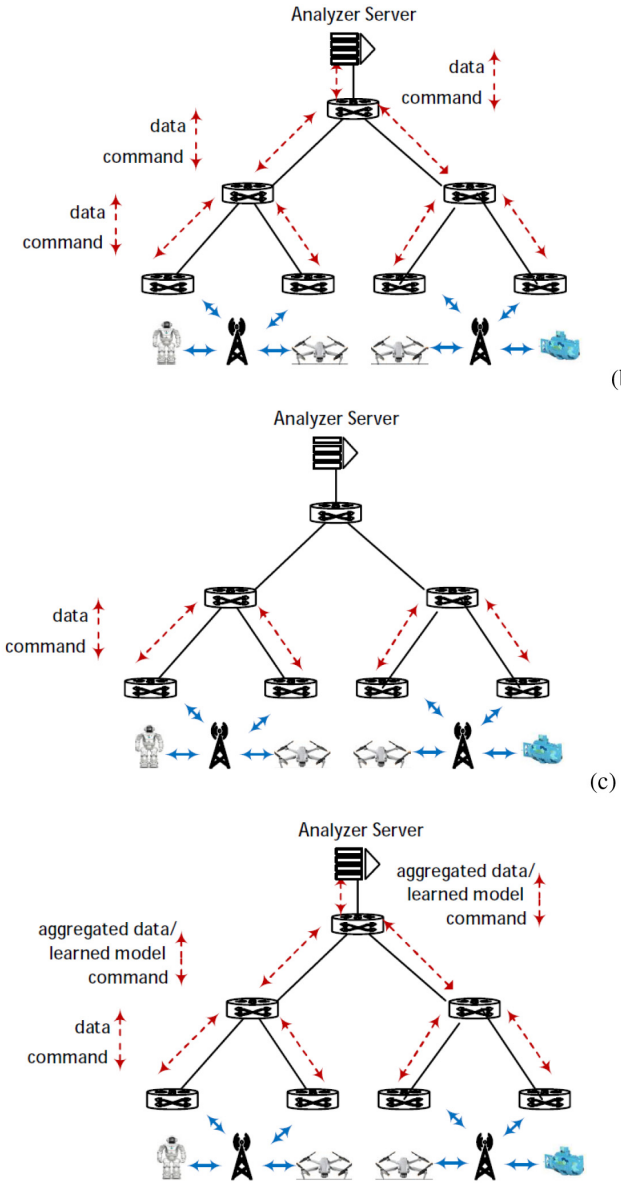

Fig. 6. (a) Performing analysis on the data in the access layer without in-network computing, (b) Performing analysis on the data by in-network computing when the collected data are sufficient to apply the analysis, (c) Performing analysis on the whole data collected in access layer leveraging in-network computing.

1) In-Network Analytics: Fig. 6a illustrates a scenario where data analytic can be performed in a network with hierarchical structure. At the access layer the data is collected from some devices (e.g., IoT devices) and is fed into a network of switches to reach to a server performing analytic, namely called analyzer server. This server can implement Machine Learning (ML) or a kind of aggregation to infer a model. Analyzer server can be a server located at edge. After the server performs analysis, it will return the respective command to the target device (e.g., actuator).

Fig. 6b illustrates the scenario where the switches are programmable and in-network computing is leveraged. The

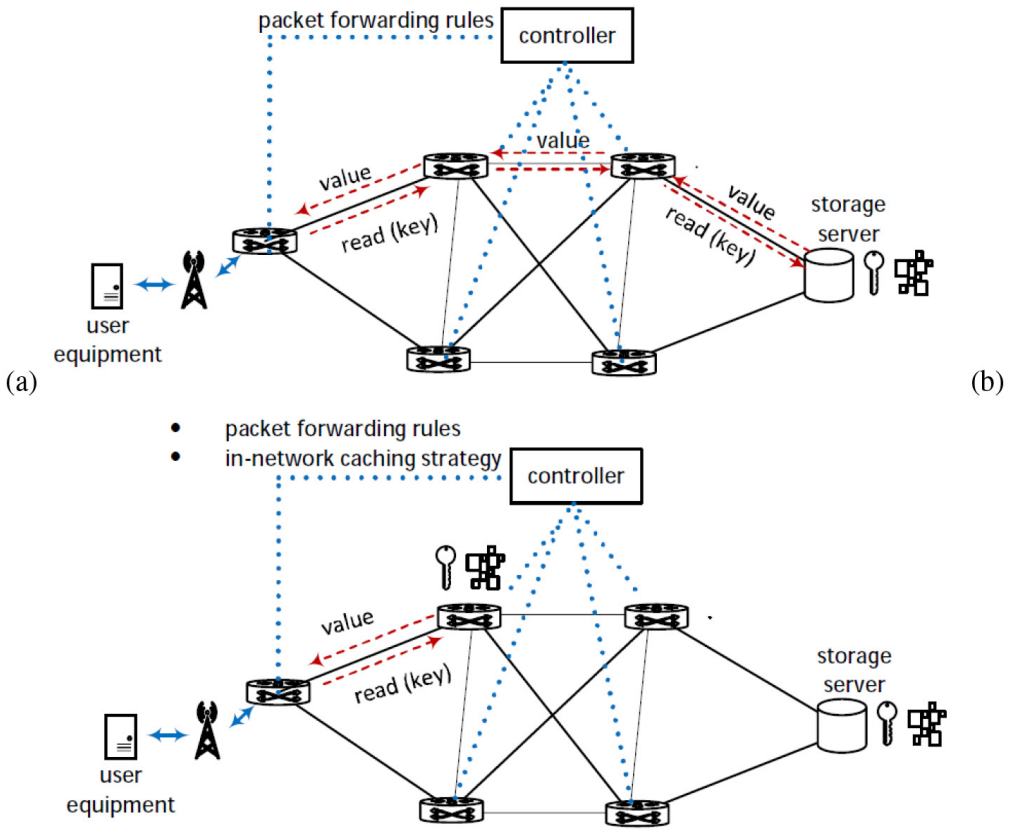

Fig. 7. (a) Caching system in general, (b) In-network caching.

scenario is for the case that sufficient data has been gathered before applying the analytic. Here, for example, at the switches in the middle level the required data has been gathered and the ML/aggregation can be applied. Let assume that the ML/aggregation method has been implemented in the middle level switches such that the switches can apply the analytic on the packet they receive. In this situation, running ML/aggregation within the network, before the data reaches analyzer server, not only terminates the traffic close to the end device and saves bandwidth at higher levels of the network, but also ensures a fast command issuance.

Fig. 6c illustrates the scenario for the case that all data collected in the access layer is required to be included in the analytic. Even, in this case the in-network computing could be effective by offloading the ML/aggregation methods to the programmable switches. The ML/aggregation is performed on the data by switches as data traverses the network. Data enters the network from the switches bounded to the access layer of the network, and makes its way up to the network, with ML/aggregation occurring at switches. A programmable switch receives data from all its children and performs the ML/aggregation operation. The learned model/aggregated data is sent by the switch to its parent switch. The analyzer server performs the final learning/aggregation and forwards the commands downward to the devices. The volume of learned model or aggregated data is less than the size of original data, thereby reducing the volume of data as it goes, instead of waiting for the data to reach the analyzer server to operate on this data. Thus, the advantage of bandwidth saving holds for this scenario as well.

2) In-Network Caching: Fig. 7a indicates a general scenario for caching when there is a storage server or original server serving contents to the users. The storage server can be provided by edge computing. User equipment can be connected to the storage server by a network of programmable switches. The items are modeled as pair of key-values. Without loss of generality, we assume the storage server is capacious to store all the items. The controller applies packet forwarding rules to the switches. In a general scenario and without in-network computing, the switches play the role of packet forwarding. When a read-request arrives to the network, the switches apply the installed forwarding rules and forward the request to the storage server. The request will go all the path until it reaches the storage server. Then, the value associated with the requested key will be turned back all the path to reach to the user.

Fig. 7b indicates the delivery scenario when we have in-network caching. The switches on the network between the user equipment and storage server are responsible for implementing on-path caching for key-value items and also routing packets using standard protocols. The packets that are supposed to use the in-network caching service might be distinguished from ordinary packets so that required processing be performed to deliver them in-network caching service. Switches have a key-value storage module implemented as match-action tables to store the hot items. The

controller is responsible for updating the switch storage with the hot items based on an in-network caching strategy. The detection of hot items can be performed based on the statistical information about items calculated by the switches and sent to the controller. When a read query arrives, the switch checks whether the cache contains the item or not. If it is a cache hit, the switch returns the associated value to the user. Otherwise, the request will follow its way toward the storage server. On the path, whenever any switch detects the key, the journey of the packet will be truncated. Thus, lower latency can be achievable by in-network caching.

3) In-Network Computing Benefits: In-network computing can provide the following benefits [15]:

(i) *High Throughput:* The network elements can handle in order of billion packets per second. For example, Tofino chip released by Barefoot supports 12.8 *Tb/s* line-rate processing. Therefore, the in-network computing paradigm provides orders of magnitude higher throughput processing capacity in comparison with the host-based solutions.

(ii) *Low Latency:* The host-based solutions suffer from inherent uncertain delay and jitter. By contrast, the network elements support sub-microsecond processing latency. As pipeline design does not access external memories in each stage, the latency is almost stable, i.e., low jitter. As we discussed through the illustrative use cases, in-network computing performs the computing inside the network and consequently, the transaction terminates within the path and detouring data to distant services is avoided. Therefore the delay as the result of the data transmission from the network elements to the end host will be saved. Indeed, in-network computing can be performed in the proximity of user and bring computation closer than servers in edge/cloud computing.

(iii) *Bandwidth Usage Reduction:* As we discussed through the illustrative use cases, the in-network computing terminates data computation on the path and before reaching to edge/cloud server. Thereby, there will be saving in bandwidth usage and the traffic congestion on the backhaul links can be avoided.

(iv) *Load Balancing:* A kind of load balancing emerges through leveraging in-network computing. The requests can be responded on the path by the network elements and before reaching end hosts. In this regard, the workload is divided between network elements and end hosts. For example, the traffic for latency-non-sensitive applications can be forwarded to the end hosts whilst the network elements can accommodate latency-sensitive applications.

(v) *Energy Efficiency:* Network elements consume less energy for performing operations. A typical programmable switch can perform billions of operations per watt usage of energy. For example, the Arista 7170 series programmable switch consumes less than 5 *W* per 100 *G* port. As another evidence, the experiment in [27] shows that millions of queries operation in the network elements will consume less than 1 *W* power. The processing capability of network elements per watt usage of energy is more efficient than the general purpose computers. Besides, as network elements are elements of the network doing packet forwarding as their default task, not high energy is consumed in idle mode. In contrast, we have the issue of high energy consumption in case of general purpose computers.

D. Categorization, Evaluation, and Comparison Criteria

In this section, we propose a set of criteria to categorize, evaluate and compare literature studies in the scope of in-network computing.

1) Proposed Categorization: At the first level, we propose to categorize the research studies based on the application of in-network computing. To clarify the in-network computing involvement, at each category of application then, we propose to organize the papers according to the specialized topics that in-network computing will be involved within the context of that category. Section I-C, gives the details about the structure of our proposed categorization. Here, to ease the reading of the survey, we provide an overview of the proposed category at the first level. We have found five categories i.e., analytics, caching, security, coordination, and technology specific applications:

(i) *In-Network Analytics:* The research in this category exploit network elements to perform analytics (e.g., machine learning, data aggregation, heavy flow detection, query processing, controlling, deep packet inspection) on the path and without the necessity of data be traversed toward end hosts to perform the analytics.

(ii) *In-Network Caching:* The research in this category leverage network elements to construct an in-network caching fabric atop of storage servers to reduce data access time. At this category of research, we found the studies that have been done in the scope of key-value store applications, as well the studies in the scope of information-centric caching.

(iii) *In-Network Security:* The research in this category perform a fraction or whole functionalities required to detect and mitigate network attacks in the network elements, in order to reduce the latency of attack mitigation and operational cost imposed by dedicated servers for the security purposes.

(iv) *In-Network Coordination:* Agreement on some data value or a sequence of operations is realized through consensus protocols in distributed systems that can be considered as a kind of coordination. In the literature, there exists also other types of coordination, e.g., lock management system, group cast communication, coordination for consistency. The research in this category, offload parts or whole of functionalities required for performing a coordination to the network elements to reduce the coordination latency.

(v) *Technology Specific Applications:* The research in this category provide a variety of in-network computing

TABLE I
CATEGORIES OF NETWORK ELEMENTS USED IN IN-NETWORK COMPUTING STUDIES

Category	Network Element
Programmable Switch/Router	(Barefoot) Tofino, BMv2, Open vSwitch, Realtek RLT, Switch-IB InfiniBand, P4BM
FPGA	NetFPGA, ZedBoard Zynq FPGA
Smart NIC	Netronome Agilio CX, Netronomr NFP, Cavium Octeon II CN

applications related to particular technologies including cloud computing, edge computing, 4G/5G/6G, and network function virtualization. The studies in this category offload parts of functionalities that is specific to a particular technology to the network elements.

2) Proposed Criteria for Evaluation and Comparison: We propose two categories of criteria to analyze, compare and evaluate the in-network computing papers: Common Criteria, Application Specific Criteria.

(i) *Common Criteria:* We propose the below criteria to analyze, compare and evaluate all possible in-network computing solutions as below:

- *In-network computation:* In-network computation is the task or computation that is performed in the network element. Indeed, logical components of a network element, e.g., match-action tables of a programmable switch which can be located in TCAM/SDRAM, Compute Logic Blocks of FPGA, programmable components of smart NIC, can be programmed through languages, e.g., P4 to carry out the in-network computation. Depending on the application, in-network computation can be a kind of data aggregation, a machine learning algorithm, traffic statistic calculation, security policy, a radio/core-network function of a mobile communication technology, a general network function, etc. This criterion gives insight to the researchers to estimate if the targeted computation is implementable in a specific network element. Furthermore, some lessons can be learned to fill the gap of research on a specific computation that has not yet be implemented on the network element.
- *Co-design:* This criterion defines if in the proposed method, network elements are used in conjunction with non-network elements (e.g., servers, controller) to perform the required computation or decision. The required computation is defined based on the context of the problem. In *in-network analytics*, performing analytics; in *in-network caching*, performing caching and replying to the requests, in *in-network security*, attack mitigation, and in *in-network coordination*, performing the coordination (e.g., consensus in consensus protocols) are the required computation. Similarly, in technology specific applications, required computation is determined in the context of the target problem. For example, in an attempt to perform LTE EPC control plane in the network, the required computation is LTE EPC control plane functionalities. Whenever, the proposed method is a co-design approach, only a fraction of the required computation is performed in the network element, otherwise the whole required computation is implemented in network element. In this regard, a non-co-design approach can be fully implemented in the network. This criterion gives insights to the researchers on the power of in-network computing in handling various problems. Furthermore, some lessons can be learned to fill the gap of the research, e.g., providing co-design approaches whenever we cope with hardware limitations of network elements.
- *Data structure:* This criterion defines the data structure of the network element that has been used in implementation. We have reported the usage of well-known data structures: bloom filter (data structure that provides capability to test whether an element is a member of a set), sketches (data structures capable of summarizing information about the network, e.g., getting traffic statistics requiring a fixed size memory), and hash table [44], [73]
- *Network element:* This criterion defines the network element that is used for evaluation. We found three categories of network elements in the literature: programmable switch/router, FPGA, and Network Interface Card (NIC). Table I illustrates the devices at each category. This criterion gives insight to the researchers to decide about the network element for the specific application they target. Furthermore, this criterion gives insights about the distribution of the usages of various network elements in the research. Accordingly, some lessons can be learned to fill the gap of studies.
- *Platform:* This criterion defines the platform used in evaluation of the method, which can be either hardware or software. Note that we consider the platform as software, when a software version of the network element (e.g., BMv2, software router, simulated switch) has been used in the evaluation.
- *Main result:* This survey provides the main results achieved by in-network computing appliance particularly from the aspect of latency, throughput, bandwidth saving, and power consumption.

(ii) *Application Specific Criteria:* We also propose some application specific criteria for comparison of the studies. Here, we provide a summary of the criteria and we refer readers to the dedicated section to each application for the details.

- In-network analysis: we consider *Simplification Technique* (e.g., quantization, precomputation) that is used to implement the analytic in the network element as a criteria. Furthermore, considering that the aim in analytics is inferring a model built over collected data, we will use criteria including *Model Complexity*, *Model Accuracy*, *Inference Speed*, and *Bandwidth Consumption* (due to data transmission) to compare the methods. We also

propose comparison for techniques to cope with hardware limitation in implementing analytics in the network, from the aspect of criteria including *Hardware Limitation Type*, *Operation of The Technique*, and *The Compromised Criteria* (due to applying the technique).

- In-network caching: The methods in this scope, provide in-network caching fabric to facilitate content/item access. We define an in-network caching fabric to be deep when the caching fabric contains hierarchy of in-network caches, otherwise we define it as shallow. We consider *In-Network Caching Fabric Type* (i.e., shallow/deep) as a criteria. Furthermore, we will use criteria including *Caching Hierarchy*, *Load Balancing in Requests Processing*, *Content/Item Access Delay*, *Available Storage At Edge of Network*, and *Bandwidth Consumption* (due to content transmission) to compare the methods.
- In-network security: Considering that the aim in these methods is attack mitigation, we will use criteria including *Modeling State of System in Attack Detection*, *Attack Detection Model*, *Attack Detection Accuracy*, and *Mitigation Latency* (including latency for detection and applying the security policy) and *Bandwidth Consumption* (due to traffic transmission in order to detect the attack) to compare the methods.
- Technology specific applications: We assess if the method has applied any *Optimization* with a particular *Objective Function* to optimize system or application related performance metrics. Furthermore, we will compare the methods in the literature with server/dedicated hardware based schemes from the aspects of advantages and disadvantages in terms of criteria including *Computing Node Type*, *Processing Cost*, *Latency/Throughput*, and *Power Consumption*. We will also discuss the advantages/disadvantages of the resource allocation techniques to cope with hybrid substrate network including both network elements and general purpose computation units, from the aspects of criteria including *Adaptability at Run Time* and *Fault Tolerance*.

In the rest of this survey, we review the different papers we found relevant to in-network computing and discuss them as per the above-mentioned categories and based on the criteria we specified earlier. Whenever, the surveyed method is a co-design approach, we will explain the reason behind that. The comparisons are supported by logical explanations as well as evidences collected in the literature.

III. In-Network Analytics

In this section we provide an overview on the existent studies that perform analytic in the network. Fig. 2, Section III illustrates the structure of this section. We give an overview of the research studies in the scope of in-network alalytics in three categories of *Data Aggregation*, *Machine Learning*, and *Other Analytics*. *Data Aggregation* gives overview of the studies that collect data from different sources and apply some aggregation functions or operations on the data, which can be regarded as a kind of analysis due to assembling an aggregated model from data. In these studies, network elements are utilized to perform data aggregation. Category of *Machine learning* gives overview of the studies that have implemented machine learning techniques in network elements. Finally, the category of *Other Analytics* covers other studies done in the scope of in-network analytics including heavy flow detection, controlling, query processing, complex event processing, and deep packet inspection. In the rest of this section we present the studies in the mentioned categories and finally we give a summary of studies and discuss the insights and lessens learned.

A. Data Aggregation

Data aggregation is a technique that combines data from different sources by applying aggregation functions or operations. In-network data aggregation sets up aggregation overlays on network elements to aggregate the data as it passes through the overlay. In comparison with a host based aggregation where the data is transmitted to a centralized host in order to be aggregated, in-network aggregation not only reduces the volume of traffic flow in the network, but also reduces aggregation time by utilizing fast processing speed of network elements. In this category of research, [12], [29], [74], [75], [76], [77] follow a similar multi-level architecture for data aggregation, commonly based on tree-structure, however they give protocols for various applications. While [29] propose a data aggregation protocol for high performance computing application, the studies in [12], [74] provide protocols for data aggregation in Map-Reduce based application. The study in [75] provides in-network aggregation for IoT application and the studies in [76], [77] propose in-network aggregation for wireless networks.

An efficient high performance computing architecture requires to assess alternative system elements to distribute the data manipulation as appropriate, rather than loading the processing of all data to a local or remote CPU. Offloading data manipulation to the network as data moves through it, frees up CPU cycles for computation, reduces communication latency, as well as the amount of data transferred over the network. To reach these aims, Graham et al. [29] offloads commonly used communication patterns in high performance computing, i.e., collective operations in Message Passing Interface (MPI) standard to the network. The authors present a hierarchical aggregation protocol to offload the operations to the network for the purpose of data aggregation. Reduction operations including *small data reduce*, *all-reduce*, and *barrier* are performed in network elements. Data enters the aggregation tree from its leaves, and moves up in the tree, while data reductions are performed at aggregation nodes. An aggregation node receives aggregation requests from all of its children and performs the aggregation operation. A data structure per collective operation at each aggregation node is used to track the progress of a collective operation. The result of the aggregation

is sent by the aggregation node to its parent in the tree structure. The root aggregation node does the final aggregation, generates the result of the aggregation operation, and forwards the result to the destinations. The protocol also support fault handling mechanism to cope with errors that may occur including transport-level errors, end-node errors, and protocol errors. The in-network computation are the collective data reduce operations that are implemented in Mellanox's SwitchIB-2 ASIC. The evaluation results indicate up to 70% improvement in latency for operations completion.

The studies in [12] and [74] perform in-network data aggregation for Map-Reduce based applications. Sapio et al. [12] has designed an aggregation system at which for each reducer, a spanning tree is constructed with the reducer as the root and including all the paths from all mappers to the reducer. Then, the network controller configures the network elements to perform the per-tree aggregation and forward the traffic through the tree structure. Each map task produces a set of key-value pairs, which is partitioned among the reducers. The partitions are transmitted to the reducer by UDP packets, at which each packet contains a preamble and a sequence of key-value pairs. The preamble defines the number of pairs and the tree ID the packet belongs to. For each tree, network elements store the keys and values in the memory. The authors propose an algorithm to be executed by each network element, according which the element aggregates the already values stored in its memory and the new values it receives in packets. The proposed algorithm also programs the network elements to forward the aggregated results to the next node towards the destination. The in-network computations, i.e., storing data, as well as the Map-Reduce specific aggregation are implemented by BMv2. For an application of Word-Count, the experiments indicate the proposed method provides a 87%-89% bandwidth saving and 84% reduction in latency.

Based on a Map-Reduce computation, the authors of [74] propose an architecture to perform in-network data aggregation. The proposed architecture consists of *switch*, *header extraction module*, *payload analyzer*, and *controller*. (i) *Switch* aggregates flows arriving from different ports and sends the aggregation results to the next hop. Furthermore it Forwards normal packets as well. (ii) *The header extraction module* investigates the packet header and will send normal packets to the forwarding module which operates based on L2/L3 information in the traditional routing way. However, in the case that packet is flagged for the purpose of aggregation, packet will be forwarded to the payload analyzer. (iii) *Payload analyzer* accepts payload formatted as Key-Value pairs and based on their lengths, it distributes these pairs to different processing engines. The processing engines aggregates values of the same key stored in a hash table. (iv) *Controller* builds an aggregation tree and configures the switches. The in-network computation is map-reduce specific aggregation that is implemented in NetFPGA-SUME development board. The authors developed the data plane in Verilog HDL and compiled into a NetFPGA-SUME development board and implemented a simple MapReduce-like system, which works in a partition/aggregation pattern. The proposed architecture has gained data reduction ratio up to 99% and reduced job completion time up to 44%.

Unlike [12] and [74] that perform aggregation based on *key*, Madureira et al. [75] performs the aggregation by *service ID* to cover various IoT services. The authors advocate a multilevel data aggregation architecture for the aggregation of data in Internet of Things (IoTs) application. Data is aggregated as it travels through the elements of the multilevel architecture. The aggregation protocol operates in the link layer (*L2*) of the network element. The packet header contains three main elements: (i) *Service ID* which identifies a set of similar data belonging to the same IoT application. (ii) *Data Counter* which defines the number of data blocks in a packet. (iii) *Type* which indicates the forwarding mechanism to be used to forward the packets. The switch in the aggregation hierarchy, receives packets from other switches on the hierarchy and aggregates their data (e.g., average) according to the *service ID* of the packets after some conditions are met. The major condition is met when the number of received data of the same *Service ID* reaches a threshold value. The authors also propose techniques to cope with loop and matrix data structures in the implementation which are not supported by the switch. The in-network computation is IoT specific aggregation that is implemented by BMv2. Simulation results with the environment of IoT devices and a fog gateway as sink, show that the proposed method is 5 times faster in terms of average delay compared to the scenario that aggregation is performed in IoT device.

Data aggregation protocols with a certain fault tolerance in a noisy multi-hop wireless network have been proposed in [76]. A primarily discussion of this study is given in [77]. Each node takes an m-bit integer as input, and the computation is started after each node collects a predefined amount of readings. The authors focus on performing divisible functions. After the network is clustered, two intra- and inter-cluster protocols are proposed. The intra-cluster protocol utilizes the coordination of cluster head to apply the function over the data within the cluster. Through applying the inter-cluster protocol, the local aggregated results are routed through cluster heads and relay nodes to the sink node. Encoding data to codewords and decoding data after receive are also included in the intra- and inter-cluster protocols. Furthermore, scheduling between clusters to perform transmission is utilized to reduce the interference due to concurrent transmissions. The efficiency of the proposed protocol is improved for identity function, and restricted type-threshold functions. The authors also analyze the complexity of the protocols. The in-network computation is divisible functions that are implemented in wireless network relay nodes. Though in comparison with other data aggregation methods this method discusses a more general function appliance in the network, the discussions are based on analysis and there is no evaluation of the proposed method.

B. Machine Learning

Machine learning algorithms are widely adopted for classifying or performing regression over incoming packets. Taking

the values of packet header fields and flow statistics as input features, these algorithms are able to learn the pattern of traffic from collected network traces and make predictions for future inputs. To apply machine learning, an unknown packet incoming to a switch has to be forwarded to a remote server where the learning algorithms run. However, the imposed delay and bandwidth consumption will be high. Studies have been carried out to leverage in-network computing to reduce learning processing time, respond earlier to the events, and terminating traffic close to the edge. While [78] implements various classification approaches including decision trees, SVM, naive Bayes, the studies in [79], [80] implement neural networks, and finally, the study in [81] implements a federated learning through network elements. In the rest of this section we provide more details about the aforementioned studies.

Implementing decision trees, SVM, and naive Bayes, and k-means on the switches have been considered by Xiong and Zilberman in [78]. Without considering the way through which the mathematical functions can be implemented in the switches, the authors utilize look up tables to store the results of calculations, and suggest some algorithms to implement the aforementioned classifications and clustering on the switches. For decision tree, in every stage of the switch, one feature is matched with all of its potential values. The result, i.e., the action, that indicates a branch happened in the tree, is encoded into a metadata field. The last stage within the pipeline calculates the final result based on the metadata fields of all features. SVM is implemented in several tables, each one indicating the status of input in relation to a hyperplane. Indeed, the key in the match-action table is the set of features of a given input, and the action is the *vote* which indicates whether the input belongs within or outside of a hyperplane. Once an input is matched against all tables (i.e., all hyperplanes), the class with the highest number of *votes* will be the classification's result. Naive Bayse is implemented by using one table per class, and features as the keys. The returned value is an integer value that indicates the probability used in naive Bayse learning. The authors also discuss different mapping of clusters and features to tables for the K-means clustering. The in-network computations are decision tree, SVM, Naive Bayse, and K-means that are implemented in BMv2 and NetFPGA SUME. The proposed method is considered to be a co-design approach since it initializes the look up tables with required mathematics within the learning. The evaluation is performed for IoT traffic classification with various classes (e.g., audio, video, etc). The latency for classification (inference) at the line rate, i.e., 2.6 μs, and the accuracy up to 94% have been reported.

A neural network implementation on network elements has been discussed in [79]. Through the experiments the authors show that the overhead of moving data is high when off-path accelerators such as GPUs or TPUs, are used to run AlexNet neural network. Extra data movement overhead will be diminished by implementing the accelerators within the network elements on the path. However, utilizing network elements to perform neural network processing, requires the splitting of the processing which will have overhead. The authors have reduced the overhead by suggesting an appropriate split mechanism. The all or part of neural network parameters are stored in SRAM of network element. A process called quantization, i.e., binarization of activation and parameters, that simpifies the operations of a fully connected layers of a neural network is utilized to reduce the number of bits required for representing a neural network's activations and parameters, as well as using simpler arithmatic operations. The in-network computation is neural network that is implemented in network processor-based SmartNIC. The processing of a single layer of neural network takes 1 *ms* in the network, while this latency is up to 12 *ms* when the processing is performed by CPU.

Simalar to [79], Lu and Lin [80] propose a method for neural network inference in the network. The authors develop a data forwarding processing system that allows packets to be cloned to the kernel of a switch for on-line inference. To provide the capability of in-switch inference a hardware called neural compute stick (NCS) is utilized. NCS is connected to a P4 switch over a USB interface, so that the cloned packets be processed and real-time inference be performed. The proposed architecture consists of two phases: (i) an offline model training phase, which uses a CNN architecture LeNet-5 to create the inference model. (ii) an online inference phase which runs the inference model within an NCS for online inference. The in-network computation is neural network that is implemented in Edge-core Wedge 100-32X switch. For a convolutional neural network based malware-classification problem, the inference time when it is performed in the network is 9 *ms*, while it is 44 *ms* when the inference is performed in a server. The accuracy is higher than 94%.

Qin et al. [81] leverages in-network computing and propose a line-speed framework for federated learning. A neural network classification with the binary weights and sign function as the activation function is performed at gateways that forwards packets from/to the devices in a network domain. A neural network with real valued-weights is stored in the control plane to re-train the classification algorithm in the gateway through performing backward propagation. After local training, gateways send the local updates to the cloud that acts as the aggregator. In order to reduce the communication cost, each gateway reports only the 1-bit sign of local updates. Then, the cloud will announce the result by a majority of voting mechanism. The in-network computation is the neural network which is implemented using BMv2. The proposed method is a co-design approach since it leverages cloud computing for aggregation and control plane for the purpose of backward propagation and updating. The evaluation with one fully connected hidden layer with 120 neurons, for the purpose of malware traffic detection, illustrates that the inference latency in the network is less than 2 *ms* for 95% of packets.

C. Other Analytics

In-network computing has been leveraged for other analytics in the applications of heavy flow detection [59], [82], [83], controlling [84], [85], query processing [86], [87], [88], complex event processing [89], and deep packet inspection [90]. In this section we provide an overview of these studies.

Heavy flows detection has benefits for many network management applications including mitigating link congestion,

detecting network attacks, scheduling of network capacity, etc. Existing heavy flow detection methods are commonly implemented on the control plane of a software define network. Thereby, having the overhead and additional delay in decision making due to frequent communication between the control and data planes. In-network computing has been leveraged to diminish this overhead. The studies [59], [83], [82] are similar in implementing counting the packets per flow in the data plane to estimate flow size. For the analysis of the counted values, a machine learning approach has been utilized in [59], while a threshold based decision has been suggested in [82], [83].

Zhang et al. [59] propose a decision tree based scheme for detecting heavy flows on the programmable data plane. The proposed method contains two steps of *offline model training* and *online inference*. In *offline model training*, the controller trains decision tree and compiles it into resources of target switches. In *online inference*, based on the decision tree, heavy flow is detected in the programmable data plane. In the pipeline, first packet's header is extracted and the appropriate information is saved in the registers. When the number of received packets of a flow reaches a threshold value, the flow will be assessed by the decision tree and tagged with a flag in the case of heavy flow detection. Accordingly, the desired actions can be followed upon the detection. The in-network computation is decision tree inference phase which is implemented using BMv2 and Flnet S9180-32X with a Barefoot Tofino 32D ASIC. The proposed scheme is a co-design approach, since the training phase is performed by controller. The detection accuracy is up to 98% and an average throughput of 9.4 Gbps has been gained in the hardware switch.

Harrison et al. [83] also considers heavy-flows detection, however they consider a network-wide detection scenario realized through a global statistic analysis. The authors divide the detection process between switches with PISA architecture and a coordinator. Incoming packets with the same key, such as a source IP address, source-destination pair, or five-tuple will be counted at edge switches. When the count for a key exceeds its local threshold defined by coordinator, the switch sends the coordinator the exceeded key and the count. The coordinator aggregates statistics from various switches and identifies the heavy flows. Furthermore, the coordinator updates the local thresholds associated with keys in the switches. Counting packets with the same key and and a threshold based comparison are in-network computation that are implemented in Barefoot Tofino. The proposed method is a co-design approach since the coordinator identifies the heavy flow and updates the the threshold values on the switches. The evaluation results illustrates up to 70% bandwidth saving in comparison with a benchmark method.

Sivaraman et al. [82] also propose an algorithm to be implemented in programmable switches to detect heavy flows. Similar to [59], a table is utilized to identify flow keys and their associated counter, at which the counter indicates the packets counts associated to the flow. Upon packet arrival, if the table lacks the count information about its corresponding flow, and there is space in the table, the new flow with an initial count value of 1 is inserted to the table. However, when the flow count information is already in the table, the corresponding flow counter will be updated. In the case that the table is full and it lacks the information of the flow, the flow entry that has the minimum counter value will be replaced in the table to indicate the statistic for new arrived flow. Packet counting of flows and heavy flow detection are in-network computations. The proposed method can detect heavy flows with accuracy of 95% while consuming less than 80 KB of memory.

Vestin et al. [84] leverage in-network computing for the application of sensor/actuator control networks. The sensors periodically generate data, which are transferred to a controller in order to be analyzed. Accordingly, the controller sends control actions to actuators. To reduce the latency as a result of the communication to the controller, the authors offloads parts of the controller functionality to the data plane of programmable switches. A history of sensor values will be cached at switches. The logical expression corresponding to controlling decision, is transformed into a Conjunctive Normal Form and is implemented in the switch tables. Accordingly, the switch will trigger the controlling decisions to the actuators. Furthermore, a proactive link repair scheme is proposed whenever there is a link failure. The in-network computation consists of data caching, processing and controlling decision that is implemented in P4 switches. The experimental results show that the proposed method reduces the sensor-actuator delay by 6 *ms* in comparison with the case that controller sends the control action.

Cesen et al. [85] focus on an ultra-low latency robot control problem at which the network composed of a robot and a controller, where the robot arm is programmed to do the well-structured repetitive tasks. There is a TCP communication between controller and the robot. Controller analyzes incoming messages from robot side, and accordingly it sends controlling commands, e.g., a stop message to the robot when the robot deviates a specific threshold position. A network element is located between the controller and the robot, that is used to forward the traffic between robot and controller. The authors argue that offloading latency-critical applications to the switch, reduces latency by bringing some controlling mechanism much closer to the robot. For the aforementioned use case, the detection of a specific robot position and accordingly sending a stop-movement command to the robot is the in-network computation that is implemented by BMv2. The evaluation results illustrates a negligible latency for controlling command issuance in order of *ns* in comparison with the latency of command issuance from controller which is in order of *ms*.

The studies in [86], [88] propose in-network query processing for respectively, the application of network telemetry and tolls computation for vehicles. Existing telemetry systems that employ stream processors incur substantial bandwidth and processing costs and can not provide efficient processing in the scales of multiple hundred million operations per second, as it is the demand in nowadays network. Supporting these demands using modern stream processors will impose high cost due to the low processing capacity per core. On the other hand, providing a query processing systems that rely only on

programmable switches will trade off expressiveness due to processing limitation in switches.

Gupta et al. [86] exploit a hybrid programmable switches and stream processor to achieve both expressiveness and scalability. The authors provide an interface to express queries for a wide range of common telemetry tasks. Each query consists of a sequence of data flow operators (e.g., filter, map, reduce, and join). Data flow operator are mapped to match-action tables in the data plane. Some operations like Join that are costly to be implemented in the data plane, are divided into a set of sub-operations. Decision is taken about the execution of each sub-operation in the data plane and ultimately joining of the results will be performed at a stream processor. To decide about the partitioning, a planner is proposed that solves an Integer Linear Program that minimizes the number of packets sent to the stream processor while considering constraints on the resources available in the switches. Teixeira et al. [87] extend the system in [86] by providing more functionalities to monitor the packet processing inside switches. The in-network computation is query operations that is implemented in Barefoot Tofino. The proposed method is a co-design approach since it utilizes stream processor in conjunction with switches to handle query operations. The evaluations shows the workload on stream processor is reduced up to 99% in comparison with benchmarks.

Jepsen et al. [88] discuss an implementation of a query system for an application of computing tolls for the vehicles on a highway. The query system receive historical data, e.g., the location and speed of vehicles, tolls imposed to a vehicle, travel time between two segments of highway. Various queries and notifications for the application are defined, e.g., toll notification to the vehicle, accident notification, queries about the imposed toll to a vehicle. A benchmark is implemented in a switch using the P4 language. P4 header with fixed-width fields are defined, among which there is a field that specifies the data type, e.g., a data for position report or data for alerting an accident. Accordingly, based on the data types, the tables and control flow that indicates the direction of the packets for processing in the pipeline are defined. The authors also provide perspective on the challenges for implementing a general stateful abstraction in P4. The in-network computation is query processing that has been implemented on BMv2 and Barefoot Tofino and the code is available online, however the authors have not provided any evaluation in the paper.

Network packets convey basic events such as sensor data, management data like intrusion-detection systems or anomaly detection. Complex Event Processing (CEP) infers higher-level knowledge, i.e., complex events, by evaluating incoming information, i.e., basic events. Traditional CEP is performed on servers or overlay networks. However, as suggested by Kohler et al. [89], through leveraging in-network computing for CEP, detouring of data streams to distant servers will be avoided, thereby reducing communication latency, and optimizing bandwidth consumption. Furthermore, high processing capabilities of networking hardware can provide an efficient CEP system. The authors show that it is feasible to express CEP operations in P4. The data plane consists of a set of end-systems that are interconnected by a set of programmable network processing elements. End-systems host event-based applications and can be either event sources or event sinks. Event sources observe basic events and disseminate them, while event sinks receive and react to complex events. The network processing elements implement forwarding of non-CEP packets, as well as CEP function, i.e., window operators, and the event detection engine. Window operators store and aggregate several last values of header fields. The event detection engine detects complex events based on a state machine implementation. The authors also present a tool to compile CEP operations to P4 code. The in-network computation consists of aggregation functions and complex event detection that is implemented in both Netronome Agilio smart-NIC and BMv2. The evaluation illustrates the outperforming in NIC, in comparison with BMv2 from the aspect of latency and throughput. For a complex event composed of two basic events, the detection latency in the range of 10 μs to 29 μs and throughput in the range of 16% to 56% for NIC have been reported.

Deep Packet Inspection (DPI), investigates payloads of the packets for discovering patterns written as regular expressions. DPI is deployed in either using dedicated appliances, i.e., middleboxes, or implemented as software at the end host. The former is expensive and difficult to manage and update, while the latter, imposes a computational burden on general purpose computers and suffer from performance fluctuation due to load on the servers. To overcome these issues, Hypolite et al. [90] utilize existing network processing hardware to implement DPI at line rates. The proposed method utilizes the Aho-Corasick algorithm in order to compile regular expression and convert it into a deterministic finite automaton which is implemented using a state transition table. End states demonstrate pattern matching, and a table is used to map end states to the set of matched patterns. The proposed method, provides stateless intra-packet and stateful inter-packet regular expression matching capabilities. The in-network computation is regular expression matching for the purpose of DPI that is implemented by Netronome NFP-6000 SmartNIC. The evaluation shows throughput gain up to 20 Gbps.

D. Summary, Comparisons, Insights and Lessons Learned

In this section, we first briefly summarize the studies have been done in the scope of in-network analytics, and then discuss the comparison, insights and the lessons learned.

1) Summary: This section gives an overview of the studies carried out in the scope of in-network analytics.

Several studies have been done that propose data aggregation in the network [12], [29], [74], [75], [76], [77]. In comparison with a host based aggregation where the data is transmitted to a centralized host in order to be aggregated, in-network data aggregation can reduce the volume of traffic flow in the network and aggregation time. To perform data aggregation a multi-level overlay is constructed across the networks. Then, Data is gathered from multiple sources and aggregation function is performed at network elements as the traffic initiated at data sources goes up through

the constructed overlay. The researchers have proposed data aggregation methods for various applications including high performance computing applications [29], Map-Reduce based applications [12], [74], IoT application [75], and wireless networks [76], [77]. Depending on the application, the aggregation function is defined: HPC specific functions, e.g., collective operations in Message Passing Interface standard [29], Map-Reduce specific functions [12], [74], IoT specific functions [12], wireless network specific function, e.g., divisible functions [74].

Unlike common machine learning techniques that require the traffic be forwarded to a remote server or host, in network application of machine learning reduces delay and bandwidth consumption. A few studies have focused on implementing machine learning in network elements. Implementation of various classification approaches including decision trees, SVM, naive Bayes, as well as k-means clustering on programmable switches has been studied in [78]. Implementation of neural networks in network elements has been considered in [79], [80]. The study in [81] implements a federated learning through network elements. Due to memory and processing limitation of network elements some simplification techniques have been applied, e.g., quantization technique [79] and simplified neural network model [81].

Other in-network analytics have also been carried out. Existing heavy flow detection methods are generally implemented on the control plane of software defined networking paradigm. Detecting heavy flows in the network elements reduces overhead and additional delay in decision making. In network implementation of heavy flow detection, based on counting the packets per flow in the data plane and a detection mechanism has been considered in [59], [82], [83]. The studies in [84], [85] exploit programmable data plane for controlling purposes. Parsing of packets, and finally controlling decision in a sensor/actuator control network utilizing switches has been investigated in [84]. A simple robot control has been offloaded to programmable switch in [85]. In-network implementation of query processing has been considered in [86], [87], [88]. Finally, network elements have been exploited for complex event processing [89], and deep packet inspection [90].

2) Comparisons, Insights and Lessons Learned: Table II compares the reviewed studies from the aspects of the contribution, methodology (in-network computation, co-design criterion, data structure of the network element used in method, simplification used in the method), and evaluation (network element, platform in simulation, main results). As it can be seen in Table II, the studies in the literature, either fully implement the required analytics in the network (the studies with the entry of 'N' for co-design) or follow a co-design approach (the studies with the entry of 'Y' for co-design) at which network elements are utilized in conjunction with servers or controllers in providing the purposed analytic.

Those studies that fully implement analytic in the network, mostly perform analytics such as key-value based data aggregation, and statistical counting-based inference which can be implemented in network elements with the existent hardware capabilities and data structures (e.g., bloom filter, sketches, hash table, cache). In contrast, those machine learning studies, demand more complex calculations that require techniques to cope with hardware limitation to be able to be implemented in the network.

- *Comparison of Techniques to Cope With Hardware Limitation:* Table III compares the utilized techniques to cope with hardware limitation. Quantization techniques as advocated in [79], [81], facilitate inference calculations by applying simplification in learning model such as utilizing binary weights or sign function as activation function in neural networks. However, by quantization, the accuracy will be trade off for the inference speed achieved through in-network computing. On the other hand, precomputation techniques like [78] precalculate the required statistics, e.g., Gaussian based likelihood calculation in Naive Bayse, and fill the required look-up tables in the network element. This technique has the potential to acquire higher accuracy in comparison with quantization technique, with providing more precise statistics. As another hardware limitation, there is also parallel processing limitation due to limited number of pipeline/stages, and match-action table entries. Packet re-circulation copes with the limitation through re-circulating the packet through the network element to apply the required processing [79]. However, the latency will be increased which might end to line-rate processing violation in high rounds of circulations. According to [79], to execute the 4096 neurons of a single layer on a switch with capability of at most 96 neurons in parallel, that would require the circulation of the packet for 43 times which prolongs the latency.

Table IV compares the fully in-network analytic methods, server/controller based analytic (which is the baseline for comparison in studies like [79], [80], [84], [85]), and co-design schemes (see Table II for co-design schemes) from the aspects of model complexity, model accuracy, inference speed, and bandwidth consumption.

- *Model Complexity:* Fully-implemented methods in the network have less capability than the server/controller based analytic methods due to hardware limitations (e.g., limited number of stages/pipelines/logical units, limited number of match-action entries, registers, and data structures), and lack of general purpose computing capabilities like float point operations (e.g., in programmable switches/routers). The co-design schemes can promote the fully in-network analytic methods by exploiting general processing computing capabilities. As an example, exploiting general purpose computers to precalculate the likelihood probabilities and injecting the calculation in the network element (in available registers) makes the implementation of complex learning models like Baysian and SVM possible [78]. As another example, handling complex queries in stream processor server, while offloading simpler query handling in the network as advocated in [86] let the implementation of complex query handling model.
- *Model Accuracy:* Fully- in-network implemented analytic can end to lower accuracy in comparison with

TABLE II
COMPARISON OF IN-NETWORK ANALYSIS. DS (DATA STRUCTURE), SIMPL. (SIMPLIFICATION), PLAT. (PLATFORM), Y (YES), N (NO), H (HADRWARE), S (SOFTWARE)

Scope	Ref.	Main Contribution	Methodology In-network Computation	Co-Design	DS	Simpl.	Evaluation Network Element	Plat.	Main Results
Data Aggregation	[29]	Enhancing collective operations in Message Passing Interface standard	Collective data reduce operations	N	-	N	Mellanox's SwitchIB-2 ASIC	H	Up to 70% latency reduction
	[12]	Proposing in-network data aggregation for Map-Reduce based applications	Storing data Map-Reduce specific aggregation	N	Hash Table	N	BMv2	S	87%-89% bandwidth saving 84% latency reduction
	[74]	Proposing an architecture to perform in-network data aggregation based on a Map-Reduce computation	Map-Reduce specific aggregation	N	-	N	NetFPGA-SUME	H	Up to 99% bandwidth saving 44% latency reduction
	[75]	Proposing a data aggregation protocol in the application of internet of Things	IoT specific aggregation	N	-	N	BMv2	S	80% latency reduction
	[76]	Proposing a data aggregation protocol with a certain fault tolerance in a noisy multihop wireless network	Divisible functions	N	-	N	Wireless network relay nodes	-	-
Machine Learning	[78]	Providing algorithms to implement machine learning methods into programmable switches.	Decision tree; SVM; Naive Bayse; K-means	Y	-	Y (Precomputation of look up table)	BMv2 NetFPGA SUME	H/S	Latency 2.6 μs Accuracy up to 94%
	[79]	Providing a method to implement neural networks on network elements.	Neural network	N	-	Y (Quantization)	Network processor-based SmartNIC	H	Up to 92% latency reduction
	[80]	Proposing an architecture for neural network inference in the network	Neural network	N	-	N	Edgecore Wedge 100-32X switch	H	79% latency reduction Accuracy higher than 94%
	[81]	Proposing a line-speed framework for federated learning	Neural network	Y	-	Y(Quantization)	BMv2	S	In-network inference latency 2 ms
Other Analytics	[59]	Proposing a decision tree based scheme for predicting heavy flows	Decision tree	Y	Hash Table	N	BMv2 Flnet S9180-32X with a Barefoot Tofino ASIC	H/S	Throughput 9.4 Gbps Accuracy 98%
	[83]	Proposing an algorithm to detect heavy-size flows, with the focus on a network-wide traffic detection	Counting packets with the same key; Threshold based comparison	Y	Hash Table	N	Barefoot Tofino	H	Up to 70% bandwidth saving
	[82]	Proposing an algorithm to be implemented in programmable switches to detect heavy flows	Packet counting of flows; Heavy flow detection	N	Hash Table	Y (Random instead of full search to find the minimum counter value)	-	-	Accuracy of 95%
	[84]	Proposing an in-network control mechanism for the application of sensors/actuators control networks	Data caching, processing; Controlling decision	N	Cache	N	P4 switch	H	6 ms Latency reduction
	[85]	Proposing a controlling method for a robot control problem	Detecting the robot position violation and sending control command to the robot	N	-	N	BMv2	S	Latency improvement in order of ms
	[86], [87]	Proposing a telemetry system that coordinates joint collection and analysis of network traffic leveraging a combination of network switches and stream processor.	Query operations	Y	Hash Table	N	Barefoot Tofino	H	Up to 99% bandwidth saving
	[88]	Proposing an implementation of a query system for an application of computing tolls for vehicles on a highway	Query processing	N	Hash Table; Sketch	N	BMv2 Barefoot Tofino	H	-
	[89]	Proposing an in-network computation for complex event processing	Aggregation functions; Complex event detection	N	-	N	Netronome Agilio NIC BMv2	H/S	Latency: 10 μs to 29 μs Throughput up to 56%
	[90]	Proposing a deep packet inspection for regular expression matching using programmable data plane	Regular expression matching	N	Hash Table	N	Netronome NFP-6000 SmartNIC	H	Up to 20 Gbps throughput

server/controller based implementation, due to probable simplifications and approximations that will be performed to implement the targeted analytic in the network element, e.g., binary weights in neural network or sign activation function instead of a more precise activation like sigmoid function [81]. Co-design approaches have the

TABLE III
COMPARISON OF TECHNIQUES FOR COPING WITH HARDWARE LIMITATION IN IMPLEMENTING MACHINE LEARNING IN THE NETWORK

Technique		Hardware Limitation	Operation	Compromised criteria
Simplification	Quantization	Processing limitations	Applying simplifications in model e.g., binary weights or simple activation function in NNs	Accuracy
	Precomputation	Processing limitations	Precomputation of required statistics	-
Packet Recirculation		Limitation in parallel processing due to limited number of pipelines/stages/MAT entries in network element	Recirculate the packet in network element to apply the requited processing	Latency

TABLE IV
COMPARISON OF IN-NETWORK ANALYTICS, SERVER/CONTROLLER-BASED ANALYTICS, AND CO-DESIGN SCHEMES

Feature	Fully in-Network Analytics	Server/Controller-Based Analytics	Co-Design Schemes
Model Complexity	Lower capability due to hardware limitation	Higher capability due to general purpose computing capabilities	Higher capability than fully in-network analytics due to exploiting general purpose computation
Model Accuracy	Can be lower accuracy due to approximations/simplifications	Higher accuracy due to general purpose computing capabilities	Can be higher than fully in-network Analytics
Inference Speed	Higher speed • Pipeline pattern and in the line rate processing at network elements • Performing inference at edge	Lower speed	Lower speed than fully in-network analytics and higher than server/controller-based analytics
Bandwidth Consumption	Lower • Terminating traffic at edge	Higher	Lower than server/controller-based • Still some traffic termination at edge Higher than fully in-network analysis • Some data will be transferred to server/controller to be processed • Some controlling signals will be transferred between server/controller and network element

potential to gain higher accuracy in comparison with fully-in-network implemented schemes, through offloading complex operations to general purpose computing units.

- *Inference Speed:* Fully in-network analytics methods have the capability to process the arrived packets with a compatible pipeline pattern and in the line rate in network elements and close to the end-device. Thus, in comparison to server/controller based analytics, faster inference speed is expected as for a single layer of neural network implemented in smart NICs, up to 92% latency reduction in inference has been reported in [79]. Co-design schemes have the potential to gain higher inference speed in comparison with server/controller-based analytics since still a fraction of computation is performed in the network.
- *Bandwidth Consumption:* Fully in-network analytic methods will save bandwidth consumption due to terminating traffic at edge, so that the traffic is not required to be transmitted to the server or controller. Bandwidth saving for a map-reduce application up to 99% has been reported in [74]. The co-design approaches that still perform some traffic processing at edge will have lower bandwidth consumption than server/controller based schemes. However, the bandwidth consumption can be higher than fully-in-network analysis since either some data will still be transferred toward server/controller to be processed or some controlling signals will be transferred between server/controller and network element. An example, is the study in [81] where traffic is transfered among aggregator on the cloud and in-network neural network implemented in an edge node, to perform a weighted aggregation to construct the global learned model in a federated learning approach. To have an efficient co-design scheme, an optimization decision is required to optimally partition the computation among the server/controller and the network element so that communication overhead between the server/controller and network element be minimized.

There are also some insights from which some lessons can be learned:

- *Aggregation With High Key Volumes:* An insight is that in-network data aggregation based on key-value data structure has been studied in [12], [74]. However, no data aggregation mechanism has been suggested for the applications that there are a large number of keys (e.g., word count applications at which large number of words appear in the text). This is critical due to memory limitation size of network elements that can not occupy any arbitrary

number of keys. A lesson learned is that more research effort is required to provide aggregation mechanisms for high key-volumes.

- *Machine Learning:* Another insight is that only four works study machine learning implementation on network elements, among which [79], [80], [81] consider neural network implementation, while only [78] considers non-neural based learning mechanisms. However, this study assumes that the calculations of mathematical functions used in the learning process have previously been set up as look up tables inside the device. A lesson learned is that more investigation is required for implementing non-neural based learning methods inside network elements. Particularly, more research is required to develop methods to implement the mathematical functions in non-neural based learning methods (e.g., Naive Bayes, SVM etc.). Another insight is that even neural network methods do simplifications (e.g., in weighting or activation function), in order to implement the learning in the network element, which can end to lower accuracy gain. More research is required to pursue the feasibility of implementing the neural networks without simplification techniques.
- *Co-design:* Co-design approaches can be particularly crucial for in-network analysis. The reason is that due to processing and memory limitations in network elements not all functions or operations might be implementable in network elements. Furthermore, not all operations or functions can be mapped to a specific network forwarding architecture like match-action tables. Those functions or operations that are infeasible to be implemented in network elements can still be kept at end host or controller. Few research provide co-design approaches in analytics application. For example, the study in [86] exploits a co-design approach at which a stream processor is utilized to implement complex operations in query processing. A lesson we can learn is that more research effort is needed to provide co-design in-network analytic methods.

IV. In-Network Caching

In this section we provide an overview on the existent studies that perform caching in the network. Fig. 2, Section IV illustrates the structure of this section. We first give an overview of the research studies in the scope of *key-value store* at which caching of highly-frequent key-value pairs are offloaded to network elements to reduce latency in serving queries for the key-value store based applications. Then, we give an overview of NDN, as a fundamental architecture in ICN that we found compatible with in-network caching. A comprehensive survey of other ICN architectures can be found in [68].

A. Key-Value Store

The operation of many Internet services, including search services, social networking, and e-commerce, depend on high-performance key-value stores. In order to have high performance key-value store to be able to response to massive requests in the data centers, scaling of storage servers is required which increases power consumption. Furthermore, services based on key-value store are usually sensitive to end-to-end latency. Leveraging in-network computing has the potential to significantly improve the performance of key-value sore system. Caching of contents in network elements saves traversals of data in network, reduces latency, and is ideal for handling frequently requests for the same information. The studies in [91], [92], [93], [94], [95] advocate a hierarchical caching system at which at high-level and closer to edge of network, in-network caching operates to speed up query processing, while on the lower level of caching, storage servers operate. The studies, however differ in the network element at which the in-network caching is implemented (e.g., FPGA, programmable switch), as well as the details of caching protocol. Furthermore, a replicated key-value store with programmable switches in the data plane has been proposed in [96], and a distributed shared memory system with in-network caching is presented in [92]. Note that the aforementioned studies perform storing data and caching functionalities as in-network computation. Furthermore, all of them are co-design approaches since the requested data is fetched from either the network element or back end storage. In the rest of this section we give more details.

A hierarchical key-value based caching system where at high level there exists a FPGA based in-network caching and at low level there exists a memcached server has been proposed by Tokusashi et al. [91]. The authors adopt a multi-core processor approach for query processing. The proposed architecture includes a Processing Element (PE)-network, several PEs, a memory-network, and memories including DRAM, SRAM, and CAM. Incoming queries are spread between PE elements and each PE processes a fraction of queries. Once a query is processed, the PE accesses the memory-network. There exists three types of memories in the memory-network: DRAM which contains the hash table bucket and data store chunks; SRAM which contains chunk information; and CAM which serves as a look up table for retrieving key-value pairs.

Upon a query arrival, the PE parses the packet to extract the key which its hash is used as a pointer to an address in the DRAM. If a key exists in DRAM, it would be considered as a hit. For a SET command that is a hit the key-value pair are updated in the DRAM. For a SET command with a new key, the PE assigns it to a chunk based on the list of free descriptors stored in the SRAM. For a GET query with a hit, a reply is returned to the client. In the case of miss however, the request is forwarded to the host memcached server, accordingly, the key and value in the cache and DRAM will be updated to include the missed key for later usages. Evaluations shows full line rate throughput (up to 13 *Mquery/s*), while having a latency of 1.1 μs and 80% better power efficiency in comparison with the case using only memcached server.

Another hierarchical caching system including programmable top-of-rack switch in the high level, and storage servers at lower level has been considered by Jin et al. [93]. The authors have proposed an in-network computing based architecture for key-value store that balances the load across

all storage servers in a rack within a data center. The authors utilize the theory discussion that by caching a specific amount of items in top of rack switches the load balancing between storage servers can be performed. To enable the caching in top of rack switches they propose an architecture which consists of a top of rack switch, a controller, and storage servers.

The switch provides on-path caching for key-value items, as well as supporting routing of packets using standard L2/L3 protocols. It has a key-value cache module to store the hottest items, as well as statistic elements to keep query frequency of each cached item, and to detect hot queries for uncached items.The controller updates the cache with hot items, through receiving statistical reports from the switch, and it decides which items to insert (evict) into (from) the cache.

Read queries are handled by the switch while write queries are forwarded to the storage servers. For the read queries, when these is a cache hit in switch, it inserts the cached value to the packet header. The statistical frequency element for the query will also be updated in the switch. When there is a cache miss, the query will be forwarded to the storage server which processes the query and replies to the client. The switch will inform the controller if it detects the missed query as a hot one so that controller decides about caching policy. The evaluation using Barefoot Tofino, illustrates up to 40% latency reduction and up to 10× throughput increment in comparison with baselines.

Unlike [93] that employs a shallow in-network caching fabric consisting of only a top-of-rack switch, a deeper in-network caching fabric, has been proposed by Liu et al. [94]. The proposed in-network caching fabric provides caching primitives for datacenter networks. Racks of servers are connected through a hierarcical switches including top of rack and core switches. Upon packet arrival to the switch, the packet will be forwarded to the network accelerator, which performs the computation. The network accelerator extracts key/value pairs and the command from the packet payload and performs related operations. For a new key/value pair writing command, the accelerator allocates space and writes the data. If the command is reading a value based on a key, a cache lookup will be done by the accelerator. For the case of a miss, the network accelerator forwards the request along its original path. For the case of a hit, the network accelerator constructs the reply and sends it to the switch. The authors discuss multi-path scenario for caching and handling failure of client, switch or a server. The authors prototype the proposed caching fabric by Cavium XPliant switches and OCTEON network accelerators. Their prototype reduces request latency by over 30% and doubles throughput in a cluster configuration.

Similar to [94], the study in [95] provides also a multi level in-network caching fabric including a top-of-rack switch and cache switches. However, in [94] load balancing in the in-network caching fabric is not guaranteed, while [95] considers load balancing. Liu et al. [95] proposes an in-network caching architecture for clustered storage systems at which load information is utilized to balance the query load among cache switches of in-network caching fabric. The proposed architecture consists of *cache controller*, *cache switches*, *storage servers*, and *client library*. (i) The *controller* decides about the cache partitions and updates the cache allocation under system reconfiguration events, including rack/switch insertion and system failures. (ii) The *cache switches* receive cache partition from controller and cache hot key-value objects. Furthermore, an in-network telemetry mechanism based on a piggyback mechanism is implemented in switches to distribute their load information to provide a guide for query distribution among caches. (iii) The *storage servers* host the key-value store. (iv) *Client library* provides facilities for applications to access the key-value store.

According to the proposed architecture, read queries on cached objects are replied by the cache switches. In the other hand, read queries for uncached objects as well as write queries are forwarded to the storage servers. The loads of the cache switches are stored in on-chip memory of top of rack switch, and accordingly a routing mechanism that considers loads of switches is proposed. The authors also provide mechanisms for cache coherency as well as handling failures for the cases of controller/link/switch failure. The evaluation shows up to 8.5× improvement in throughput in comparison with the case that no caching is performed.

Jin et al. [96] propose a replicated key-value store architecture that contains a *data plane* and *control plane*. (i) The *data plane* constructs a replicated, in-network key-value store, and manages read/write queries. Distribution of the key-value store over multiple switches in the data plane is done using a hashing based mechanism. When a switch receives a packet with read/write operation, it performs the required operation and updates the destination IP either to the next chain node (e.g., a miss occurrence), or to the client IP (e.g., hit occurrence). To solve the out of order arrival of packets, sequence numbers are used to serialize write queries. (ii) The *controller plane* manages switch tables and registers, as well as reconfiguration of the system due to switch failures. Switch failures, are handled in two steps of fast fail-over and failure recovery. In fast fail-over, the controller reconfigures the network to resume serving queries with the remaining switch nodes. In failure recovery, the controller adds repaired switches as new replication nodes. As failure recovery needs to copy state to the new replicas, it takes longer than fast fail-over. The evaluation shows up to $10^5\times$ throughput enhancement in comparison with a server-based solution like Zookeeper.

Wang et al. [92] present a rack-scale distributed shared memory system with in-network caching and cache coherence management. The whole architecture consists of of a set of *memory nodes*, a *top-of-rack switch*, and a *shadow node*. (i) Each *memory node* consists of a global memory to store the blocks. Furthermore, it has an application thread component that execute application logic and access global memory via write/read interfaces. It also has the cache agent component that performs transferring of cached data in the memory. (ii) In addition to routing normal packets using standard L2/L3 protocols, the *switch* is responsible for storing hot blocks, and executing part of the cache coherence protocol including serializing and multicasting of the requests. Specifically, it manages the lock to access the shared data. (iii) The *shadow node* helps migrating the ownership of cache blocks between the switch and memory nodes. Evaluation

shows up to 4.2×, 2.3× and 2× throughput speedup over distributed shared memories on key-value store, graph engine and transaction processing workloads, respectively.

B. Information-Centric Caching

The evolution of Internet usage from a host-centric communication model to a model based on the interest in accessing information, irrespective of its physical location, introduced the concept of Information Centric Networking (ICN). According to ICN, the information is requested by a naming mechanism and can be retrieved from nodes in the network irrespective of their physical location. Though the concept was introduced much prior than in-network computing, we see ICN architecture consistent with in-network caching. Particularly, the fundamental architecture of Named Data Networking (NDN also known as Content Centric Networking) introduces *content routers* that provide a content store to cache the information in the network. Here, we provide an overview of NDN, as a fundamental architecture in ICN which we found it compatible with in-network caching. We refer interested readers to [68] for a comprehensive survey of information-centric networking architectures.

In NDN [97], subscribers request information objects via INTEREST messages, which the reply will be some DATA messages. Messages are forwarded by Content Routers (CRs) that store three data structures: (i) the Forwarding Information Base (FIB) to map information names to the output interfaces in order to forward INTEREST messages, (ii) the Pending Interest Table (PIT) to store INTEREST messages that are expecting for DATA messages, (iii) the Content Store (CS) to cache information objects that have passed through the CR.

When an INTEREST arrives to a CR, CS is looked for an information object whose name matches the requested prefix. When hit occurs, the result is sent back through the incoming interface in a DATA message. Otherwise, a match will be performed in FIB in order to determine the output interface for message forwarding. Then, the INTEREST's incoming interface will be stored in the PIT and the INTEREST will be forwarded to the CR determined by FIB.

When a DATA message arrives to a CR, the information object is stored in CS and a match is performed in PIT to find the interfaces though which the DATA message is forwarded.

C. Summary, Comparisons, Insights and Lessons Learned

In this section, we first briefly summarize the studies have been done in the scope of in-network caching, and then discuss the comparisons, insights and the lessons learned.

1) Summary: This section gives an overview of the studies done in the scope of in-network caching. The *in-network computation* in this category of research, is the caching Key-Values or content and processing the arrived queries to the network element. All the surveyed in-network caching methods are regarded as co-design approaches as the caching service is served with a conjunction of in-network caching fabric and back-end storage servers.

Key-value store deployments in the data centers, experience high latency to respond the queries. In-network caching saves traversals of data in network, reduces latency and increases throughput. The studies in [91], [92], [93], [94], [95] construct an in-network caching fabric atop of storage servers, to process the queries. While [91] develops the in-network caching fabric in FPGA, [92], [93], [94], [95] utilize programmable switch for the in-network caching fabric. The in-network fabric caching is shallow in [91], [92], [93] and includes a top-of-rack switch or an FPGA, while it is deeper in [94], [95] and includes a hierarchy of switches. Generally, whenever, the query request hits in the in-network caching fabric, the requested value will be returned. On the other hand, the requests with miss in in-network caching fabric will be forwarded toward storage servers. The studies in [91], [92], [93], [94], [95] are different in the details of caching protocol. Furthermore, in the category of key-value store, a replicated key-value store with programmable switches in the data plane has been proposed in [96], and a distributed shared memory system with in-network caching is presented in [92].

The concept of ICN, at which the information or data is directly requested by a naming mechanism and can be retrieved from nodes in the network irrespective of their physical location, seems be consistent with in-network caching. Particularly, the fundamental architecture of Named Data Networking provides content stores at *content routers* to cache the information in the network. A comprehensive survey of ICN architectures can be found in [68].

2) Comparisons, Insights and Lessons Learned: Table V compares the reviewed studies from the aspects of the contribution, methodology (co-design criterion, in-network caching fabric), and evaluation (network element, platform in simulation, main results). As it can be seen in Table V, the studies in the literature, either provide a shallow in-network caching fabric or deep in-network caching fabric. Table VI gives a comparison among shallow and deep in-network caching schemes, as well as server-based caching schemes at which either an origin server at cloud or a storage server at network stores the contents. Note that the case of caching at cloud or a storage server is a baseline for comparison in many studies, e.g., [91], [94], [96].

- *Cache Hierarchy:* Shallow in-network caching schemes construct two levels of hierarchies including in-network cache element and origin server (also called as storage server). Deep in-network caching schemes construct a hierarchy including more than two levels: an in-network caching fabric with a hierarchy of in-network cache elements and origin server. In the other hand, cloud-based caching schemes provide a one level of caching which is origin server located at cloud.
- *Load Balancing:* In server-based caching schemes, all requests will be replied by a caching server and there is no opportunity for load balancing. In contrast, shallow in-network caching schemes provide the capability of load balancing as the requests will be processed either by in-network caching element or origin/storage server. On the other hand, in comparison with shallow in-network caching schemes, e.g., [92], [93] load balancing will be boosted in deep in-network caching schemes, e.g., [94], [95], [96] due to offering various dimension

TABLE V
COMPARISON OF IN-NETWORK CACHING STUDIES. THESE STUDIES PERFORM STORING DATA AND CACHING FUNCTIONALITIES AS IN-NETWORK COMPUTATION

Scope	Ref.	Main Contribution	Methodology		Evaluation		
			Co-Design	IN-Caching Fabric	Network Element	Plat.	Main Results
Key-Value Store	[91]	proposing an in-network caching system for a memcached server	Y	Shallow	NetFPGA SUME	H	Up to 13 $Mquery/s$ throughput Latency of 1.1 μs 80% Improvement in power efficiency
	[93]	proposing key-value store architecture that leverages in-network caching to provide dynamic load balancing across all storage servers located in a rack	Y	Shallow	Barefoot Tofino	H	Up to 40% Latency Reduction Up to $10\times$ throughput increment
	[94]	Proposing an in-network caching method for racks of servers connected through switches in a tree structure.	Y	Deep	Cavium XPliant Switch	H	Over 30% Latency reduction $2\times$ Throughput increment
	[95]	Proposing an in-network caching mechanism to provide caching for large scale storage systems with the characteristic of balanced load among caches	Y	Deep	Barefoot Tofino	H	Up to $8.5\times$ throughput increament
	[96]	Proposing a replicated key-value store with programmable switches	Y	Deep	Barefoot Tofino	H	Up to $10^5\times$ throughput increment
	[92]	Proposing a rack-scale distributed shared memory system with in-network cache coherence	Y	Shallow	Barefoot Tofino	H	Up to $4.2\times$ throughput increment
ICN	[97]	Proposing a data oriented network architecture which enables the in-network caching in content routers	Y	Deep	ICN caching nodes	H	-

TABLE VI
COMPARISON OF SHALLOW/DEEP IN-NETWORK CACHING AND SERVER-BASED CACHING

Feature	Shallow In-Network Caching Schemes	Deep In-Network Caching Schemes	Server-Based Caching Schemes
Caching Hierarchy	Two-Levels: In-network cache & Origin/storage server	More than two-levels: Hierarchies in in-network caching fabric & Origin/storage server	One level: Origin/storage server
Load Balancing	Yes • Balancing load between origin/storage server and in-network cache	Yes (More than Shallow in-network caching) • Balancing load between origin/storage server and in-network caching fabric • Balancing load among caches in in-network caching fabric	No
Content/Item Access Delay	Lower latency than server-based scheme due to: • Providing content/item at edge closer to end-device • Load balancing	Lower latency than shallow-in-network caching schemes due to: • Providing content/item at multiple in-network caches at edge close to end-device • Load balancing in in-network cache fabric	Higher latency than in-network caching schemes due to: • Availability of contents only at origin server located at cloud • Not capability of load balancing
Storage At Edge	Very limited in the scale of caching data structure in network element, MATs, and registers	More than shallow-in-network caching, up to maximum storage capacity of in-network caching fabric	Not available
Bandwidth Consumption	Lower than server-based scheme due to returning content/item from network element at edge and colse to end-device	Lower than shallow in-network caching due to serving more requests in in-network caching fabric	Higher due to returning the content/item from origin server located at cloud

of load balancing among in-network caching elements in in-network caching fabrics as well as between in-network caching fabric and origin/storage server.

- *Content/Item Access Delay:* Deep in-network caching schemes which provide access to contents in multiple in-network caches close to the edge as well as high degree of load balancing, can offer the lowest experienced delay for users. On the other hand, the server-based caching schemes which only provide content access at origin or storage server and do not support load balancing will experience high delay due to both long distance and congestion at core network. Over 30% reduction in latency has been reported in [94]. Accordingly, throughput enhancement is expected as up to $10^5\times$ throughput

TABLE VII
COMPARISON OF TECHNIQUES FOR COPING WITH STORAGE LIMITATION IN IMPLEMENTING IN-NETWORK CACHING

Technique	Operation	Advantages	Disadvantages
Key-Value based caching	Key-Value centric caching	Simplifying content/item storing and retrieval with key-value based approach	Small size of key-values due to storage capacity limitation e.g., TCAM and SDRAM size limitations
Deep in-network caching fabric	Exploiting multiple in-network caches (commonly has been organized as hierarchy in the literature)	• Increasing storage for caching maximum up to the capacity of in-network caching fabric • Avoiding single point of failure	• More complex controlling scheme in content/item storing and retrieval • Requiring consistency mechanisms to keep consistency among in-network caches

enhancement in query handling in comparison with a server-based key-value store solution has been reported in [96].

- *Storage at Edge:* Shallow in-network caching schemes provide very limited storage through logic units, match-action tables, and registers (e.g., order of tens/hundreds of megabytes in programmable switches). In contrast, deep in-network caching schemes will provide a storage capacity which is accumulated of multiple storage capacity of network elements in in-network caching fabric layer. Server-based caching schemes provide no storage at edge.
- *Bandwidth Consumption:* Server-based caching schemes consume considerable bandwidth in downlink due to data streaming from servers usually located at cloud. In contrast, shallow in-network caching schemes reduce bandwidth consumption by providing the content at edge. Deep in-network caching schemes can serve more requests through providing more storage capacity at in-network caching fabric which ends to higher bandwidth saving in comparison with shallow schemes.

Table VII compares the techniques in the literature to cope with storage limitation in implementing in-network caching. There are two techniques: Key-value based caching and utilizing deep in-network caching fabric. Key-value based caching, e.g., [91], [92], [93] focuses on a key-value based retrieval system which organizes the data based on small sizes of key-value pairs compatible with match-action tables. The advantage is compatibility, and the simplicity in content/item storing and retrieval, while the disadvantage is that small number of key-value pairs can be stored in a network element due to storage capacity limitation, e.g., TCAM and SDRAM size limitations. In contrast, technique of deep in-network caching fabric exploits multiple network elements in in-network caching fabric (commonly the caches has been organized as hierarchy in the literature). The studies, e.g., [94], [95], [96] as well as ICN, advocate this technique. The advantage is that the storage of in-network caching will be extended up to the capacity of caching fabric. Furthermore, the fault tolerance will be boosted by exploiting multiple network elements in the fabric. The disadvantages however, is that more complex storing and retrieval algorithms are required. Furthermore, consistency mechanisms to keep consistency of data in read/write operations among in-network caches is required.

There are also some insights from which some lessons can be learned:

- *Network Element in Caching:* An insight is that most studies in the scope of in-network caching have implemented the in-network caching with programmable switch ASIC (See Table V). An FPGA-driven implementation is followed in [91]. However the proposed method in [91] is not scalable since, it only provides caching fabric for a memcached server. There exist large number of servers in real systems, and considering that there is scarcity of memory in an FPGA, a single FPGA is not sufficient to provide load balancing among large number of servers. A learned lesson is that more research effort is required to provide in-network caching fabric based on FPGA or NICs. Furthermore, there is no research that compares an FPGA or NIC based in-network caching fabric with switch programmable ASIC based in-network caching fabric from performance criteria including latency and throughput. Thus, another lesson we learn is that research is required for the comparison between the various types of in-network caching fabric.
- *Load Balancing in In-network Caching Fabric:* Considering that load balancing in in-network caching fabric is quite important to ensure performance for the whole caching system, another insight is that load balancing in the in-network caching fabric is only discussed by [95]. In this study, the load telemetry information collected from switches in the in-network caching fabric, is used by query routing to ensure that the load between the cache switches is balanced. However, in-network telemetry mechanism applied by switches has communication overhead and consumes bandwidth for load information distribution. We learn the lesson that more research effort is required to provide load balancing in the in-network caching fabric in an efficient manner.

V. IN-NETWORK SECURITY

In this section we provide a review of research studies in the scope of in-network security. Fig. 2, Section V illustrates the structure of this section. We first review the studies in the scope of Distributed Denial of Service (DDoS) Attack Mitigation, then we follow with the studies that provide firewall solutions. Finally, we give an overview for other in-network security applications.

A. DDoS Attack Mitigation

Network environments are constantly plagued by DDoS attacks under the control of malicious actors. To mitigate DDoS attacks, scrubbing services are employed to handle the attack in the cloud. However, this mechanism results in rerouting of traffic, additional latency and higher cost for the operator in order to provide resources to handle the attack. Also, there is a risk of leaking user-related private information, when scrubbing is deployed in the cloud. To overcome these issues, in-network DDoS attack mitigation rely on network elements to analyze packet samples or flow records, to perform attack detection and mitigation in the network.

Several studies have provided mitigation mechanism for flooding attacks. Theses include SYN flooding [61], [98], TCP flooding [99], and link flooding [100]. SYN/DNS anti-spoofing is the focus of [60]. The study in [101], [102] focus on volumetric DDoS attacks. The study in [103] present an in-network defense architecture for AR-DDoS attack. A more general view of attack mitigation that includes several DDoS attacks have been presented in [104], [105], [106]. Considering the source of traffic, a proactive approach of source address validation has been considered in [107], while the study in [108], [109], [110] detects and drops spoofed traffic. Some specific cases have also been considered. The study in [111] provides a secure duplicate address detection against DoS attack. Finally, the study in [112] consider the use case of offering DDoS Protection services to universities and data centers downstream. The aforementioned studies offload either a fraction or whole of detection and/or mitigation mechanism in network elements. Below we provide a review of theses studies.

1) Flooding: SYN-specific defense mechanisms that are commonly deployed as SYN proxy, use SYN cookies or SYN authentication to mitigate SYN flood attacks. Scholz et al. [98] discuss the benefits of implementing SYN cookies and SYN authentication strategies using the P4 over data plane compared to software based packet processing with kernel-bypass. Packets are parsed in the targets and will be forwarded by L2 forwarding rules implemented by match action pipeline. The calculations for the SYN cookies and SYN authentication strategies including cookie calculations and whitelisting are implemented in the targets: (i) Some cookie-related functionalities, e.g., generating a timestamp, cryptographic hash calculation are implemented in the P4 target. (ii) The authors discuss two options to implement whitelisting: First, the data plane informs the control plane of a flow/IP address to be whitelisted and the control plane will insert an entry to the table. Second, a Bloom filter data structure is utilized for whitelisting. The in-network computation, i.e., SYN cookies and SYN authentication strategies are implemented on multiple P4 targets: T4P4S [113], a DPDK-based P4 software target running on commercial off-the-shelf hardware; the NFP-4000 Agilio SmartNIC NPU, and the NetFPGA SUME. The evaluation shows the capability of SYN flood processing up to roughly 13 Mpps in NetFPGA.

SYN flooding attack mitigation has also been considered by Lin et al. [61]. In the attack, large numbers of ACK packets with fake source IP address are sent to the server, accordingly, the server responses SYN/ACK packets to the original source. However, the server does not receive corresponding ACK/FIN packets. According to the proposed method, the number of SYN/ACK and ACK/FIN packets are counted at the switch nearest to the server, which the ratio of the counted values determines the attack. In the case of anomaly detection the traffic of the malicious source IP will be dropped and the controller will be informed. The proposed in-network attack mitigation reduces the traffic volume on the SDN controller. Furthermore, the authors propose some merging mechanism to merge the rules in the forwarding tables to reduce the cost of memory consumption. The in-network computation consists of detecting SYN flooding attack based on the ratio of SYN/ACK and ACK/FIN packets and dropping the packets from malicious source IP, that is implemented by BMv2. In comparison with the case that controller polls switches for gathering information and performing detection, the proposed in-network detection method reduces the volume of traffic up to 4000 bytes/s.

Musumeci et al. [99] propose a machine learning based attack detection mechanism with the focus on TCP flood attack. Traffic information from the P4 switch are periodically collected and analyzed by a flood detection module to detect the attack. The analysis is performed through a machine learning classifier. Considering a time window for gathering the information, some features are utilized for classification: average size of packets in time window; the percentage of TCP packets; the percentage of UDP packets; TCP/UDP ratio; the percentage of TCP packets with an active SYN flag. The authors also propose that elaborating traffic features can be offloaded from the attack detection module to the P4 switches. To perform the offloading, the proposed method exploits the potential of stateful data planes enabled by P4 language to implement packet mirroring, header mirroring and metadata extraction inside P4 switches. The in-network computation, i.e., traffic feature extraction is implemented by BMv2. The proposed method is a co-design approach since a TCP flood detection module implemented on non-network element, collaborates with network elements to detect TCP flood attack. The evaluation with SVM and Random Forest, illustrates the detection accuracy over 98%. Furthermore, the P4 switch can extract features in around 110 μs, compared with roughly 15 seconds which is required by a server-based features extraction module.

In-network implementation of link-flooding mitigation not only can detect suspicious traffic at any location/time, but also omits the necessity of a centralized controller for deploying new configurations. Applying such centralized reconfiguration takes long time which the time wast can be used by the attackers to change their strategy. In this regard, Xing et al. [100] propose a method to detect and mitigate link-flooding attacks in the network through the implementation of some boosters in the switches. In the default mode, routing is done according to the optimal policy computed by the controller. Upon detecting an attack however, an alarm is propagated through the network and the attack mitigation boosters will be activated in the switches. To minimize the disturbance to normal

traffic, the attack mitigation boosters reroutes malicious flows while the normal flows will still be routed through the original optimal route as determined by the controller. In-network computation, i.e., link-flooding detection and mitigation are implemented by BMv2. The evaluation shows that proposed method has increased the throughput of normal user flows by 80% in comparison with the case that centralized SDN controller reconfigures the network.

Afek et al. [60] focus on SYN and DNS anti-spoofing for DDoS attack. Two different SYN anti-spoofing, i.e., HTTP redirect and TCP reset, and one DNS anti-spoofing method are implemented using OpenFlow and P4 target. The main approach is a communication protocol based on cookies, i.e., a SYN-cookie based approach. Upon arrival of SYN packet, an ACK message containing a hash generated cookie is sent in response. The client is authenticated only when it responds with the correct cookie. The SYN-cookie methods are converted into primitive steps, so that each primitive step is implemented as an action in the SDN data plane. By utilizing data plane some authentication tasks, e.g., SYN cookies generation, is done without communication with the controller. To cope with the TCAM size limitation in the switches, the authors propose a method to distribute the complexity of the proposed solution over several switches. In-network computations composed of SYN and DNS anti-spoofing functions based on primitives of SYN-cookie methods, which are implemented in Open vSwitch 2.3.1. The evaluation shows that the proposed mitigation method can successfully reply to Http requests up to attack rate of 206 *Kpps* and keep the throughput up to 278 *Kpps*.

The studies in [101], [102] focus on volumetric DDoS attacks at which a large number of hosts converge traffic to one or few victims. The attack detection has been considered in [101], while the study in [102] propose a mitigation method. When attack occurs the distributions of the source and destination IP addresses deviates from the legitimate pattern. Lapolli et al. [101] measure such deviation through Shannon entropy analysis. Indeed, it is expected that in the case of attack, the entropy of source IP addresses increase and the entropy of destination IP addresses decrease. The authors propose an anomaly detection based on entropy estimation implemented by P4. The proposed method, consists of three steps: (i) For consecutive arrived packets in an observation windows, the entropies of IP addresses are estimated. (ii) At the end of an observation window, based on the central tendency and dispersion of the recent entropy values, the legitimate traffic is modeled. (iii) Threshold for attack detection are calculated, according which the attack will be detected. The in-network computations composed of Shannon entropy estimation, calculation of statistical characteristics of legitimate traffic, and threshold based attack detection, which are implemented by BMv2. The evaluation results show that the proposed method can detect DDoS attacks with accuracy 98.2% and latency 250 *ms*.

Using the entropy based analysis proposed in [101], Gonzalez et al. [102] propose an in-network pushback mechanism to mitigate volumetric DDoS attacks. The proposed mechanism removes the control plane from the critical path in order to speed up the defense. In the proposed method, first, an attack is detected through the entropy analysis of the IP addresses of packet sources. Then, upon the attack detection, forwarding device next to the victim, gives alert to the upstream forwarding devices. Accordingly, these devices, will filter packets of suspect flows, and if they also detected an attack, they will alert their upstream forwarding devices. The process will be repeated to confine the malicious traffic close to its source. The in-network computations, i.e., detection and mitigation of volumetric DDoS attacks are implemented by BMv2. The detection accuracy up to 94% and the latency of 0.1 *ms* for the reaction to the attack (from the start of attack), have been reported.

2) Amplified Reflection DDoS Attack (AR-DDoS): In AR-DDoS attacks, the connectionless nature of the UDP protocol is misused by an attacker to send spoofed requests to a server on the Internet, which responds with amplified replies to a victim. Khooi et al. [103] propose a defense architecture against AR-DDoS attack which does not depend on any scrubbing server, while provides much faster detection and mitigation of attacks. The proposed architecture deploys stateful programmable routers at the border (i.e., peering side) and access side (customer facing) of an ISP network in order to track the counts of requests/responses in a given protocol. Count tracking of requests/responses are implemented using count-Min Sketches. Then, a distributed protocol is proposed to be employed in the border and the access routers, to reach a consensus about a possible attack based on the counts-values. Finally, an access control list is developed at each border router to indicate the IP addresses of the abused servers and to ban the malicious traffic. In-network computation consist of detection and mitigation of AR-DDoS attack that is implemented by BMv2. Evaluations show that the proposed method is capable to detect and identify attacks with 99.8% accuracy in the data plane.

3) Security of Source of the Traffic: The studies in [107], [108], [109], [110] consider the security for the source of the traffic. While [107] advocates a proactive strategy for source address validation, the studies in [108], [109], [110] detect and drop spoofed traffic in a reactive manner.

Tag-based solutions for source address validation, have some drawbacks including insecure key negotiation, heavy encryption algorithms with computational overheads for routers, and using non-standard headers. To overcome these problems, Yang et al. [107] employ a secure key negotiation mechanism that can be implemented in programmable routers. The proposed method combines Elliptic Curve Diffie-Hellman Ephemeral key agreement and resource public key infrastructure to enable secure key negotiation and defeat Man-in-the-Middle attacks. Based on negotiated keys, the authors design an in-network tag generation algorithm that maps source addresses to the pseudo-random tags. The proposed tag generation method inserts the tags into appropriate packet header fields such that compatibility with standard headers be kept. The in-network computation, i.e., tag generation is implemented in commercial P4 switches.The experimental results show that the proposed method saves the bandwidth up to 200 kbps by filtering the spoofed packets.

To detect and drop spoofed packets with forged source IP address, Gondaliya et al. [108] advocate the deployment of anti-spoofing mechanism on the intermediate switches instead of deploying the mechanisms on the source or destination nodes. The authors give details and analyze P4-based implementation of several anti-spoofing mechanisms including: network ingress filtering, spoofing prevention method, variants of reverse path forwarding, and SAVI. The authors have explained the match-action table for implementing each of the anti-spoofing mechanisms. The in-network computations, i.e., anti-spoofing mechanisms are implemented on NetFPGA SUME hardware. According to the evaluations, for a packet generation rate of 8.5 Gbps and spoofed packet ratio of 12.5%, the anti-spoofing mechanisms has gained throughput of roughly 7.5 Gbps.

Similar to [108], the authors in [109], [110] also focus on filtering spoofed traffic. The authors advocate an Hop Count Filtering (HCF) defense that can filter spoofed IP traffic with an IP-to-Hop-Count (IP2HC) mapping table. Instead of applying HCF mechanism in end hosts, they propose an architecture to integrate the HCF mechanism within programmable switches. This will end to the earlier recognition of spoofed traffic whilst saving network bandwidth resources. The proposed architecture includes two planes of data and control respectively called as cache and mirror. The data plane serves most active and legitimate IPs, while the control plane manages the remaining IPs, stores the IP2HC mapping table, and updates the state of system to adapt to network dynamics. The data plane runs three modules: (i) an IP2HC module in order to validate the packets (ii) a TCP session monitoring module to track the legitimate hop-counts and update the hop-count values, (iii) a statistic module to calculate statistics about legitimate/spoofed IP. When the hop-count checking of a packet is successful, the corresponding statistic information is updated and the packet will be forwarded. However, when the checking fails, the packet will be dropped and the spoofed-packet counter is increased. The value of spoofed-packet counter is reported to the control plane periodically.

To overcome the memory limitation of switches, the control plane keep the global view of the traffic. It utilizes a binary tree data structure to aggregate the information about IPs, and it maintains a global IP2HC mapping table. Furthermore a spoofed-packet counter counts the number of packet checking failures to be used to adjust the state of system. The in-network computations composed of IP2HC inspecting, TCP session monitoring, and calculating legitimate/spoofed IP hit statistics, which are implemented in hardware Tofino switch. The proposed method is a co-design approach because the control plane that is implemented in a non-network element handles a fraction of IPs, maintains the global IP2HC mapping table, and performs required updating in data plane. For spoofed traffic whose hop count distribution follows a Gaussian distribution, the proposed method prevents spoofed traffic from entering the network host and saves bandwidth roughly 200 *Mbps*.

3) DDoS Attack Mitigation in General: A more general view of attack mitigation that covers several DDoS attacks have been presented in [104], [105], [106]. Friday et al. [104] employ programmable switch at the edge of the network, to conduct analysis on the traffic prior to the traffic interfaces with the network's internal devices. The traffic analysis can be performed at switches without overhead imposed by controller intervention. To capture the excessive SYN requests and consumption of the server's connections, several functionalities and statistics are implemented in the switches: (i) a kind of signature matching on ingress SYN packets; (ii) signature counts calculated using the Bloom filter; (iii) the number of SYN requests per a predefined time window; (iv) interarrival times of TCP connections; (v) the maximum number of consecutive sessions constructed in a time window, without already-existed sessions be closed. Once the aforementioned data is gathered by the switch data plane, the controller collects them for traffic distribution formulation and accordingly calculating a threshold. The threshold will be sent to the switch according which the switch compares its statistic, detects the attack and drops the malicious traffic. The in-network computation consists of statistic calculation (about signature, SYN requests, TCP connection, and session establishment), attack detection, and drop malicious traffic, which are implemented by BMv2. The proposed method is a co-design approach since controller estimates traffic distribution and calculates the thresholds for attack detection. For DDoS and volumetric types of attacks, by applying the proposed method only up to 3.5% of clients experience negligible delays. Furthermore, for SYN flood the detection latency is below 0.25 seconds.

Another DDoS attack mitigation in a general manner has been presented in [105]. Based on the language NetCor [114], three classes of defense primitives are defined: (i) *monitors* which collect statistics over the network traffic, (ii) *actions* which specify the defense decisions taken on a particular kind of packets, and (iii) *branches* which express the control flow of the defense. Each class of primitives can consist of several primitives. For example, for the *actions* primitives of *drop*, *pass*, *puzzle*, *log* can be defined. Then, the amount of resources required for the primitives on the switch and/or server are determined. According to the analysis results, the proposed method divides the required steps of a primitive between server and switches and decides about usage of resources in the switches, e.g., match-action tables, registers.

As the next step, a graph structure is constructed for the defense policy, at which the nodes are the defense primitives and the edges denote the flow of the traffic. Based on the analysis about the resource usage of the primitives, the primitives execution are mapped to the stages of the switches. Considering the SRAM and ALU constrints, the mapping problem is formulated as an integer linear programming optimization which maximizes the computation performed in switches. The authors also discuss how the proposed method handles dynamic attacks at run time. The in-network computation, i.e., collecting statistic of packets and defence operation are implemented on Barefoot Tofino. The proposed method is a co-design approach as a combination of switch and server are used to implement required primitives. The evaluation which is performed on SYN flood, DNS amplification, and HTTP and UDP flood attack, illustrates quickly restoring the throughput of legitimate traffic flows and restoring bandwidth roughly 20 Gbps though filtering attack UDP packets. Compared with

middlebox and NFV systems, which requires tens of microseconds, the packets can be processed within hunderds of nano-seconds.

In comparison with [104], [105], Liu et al. [106] argue for a broader-spectrum detection which covers 16 volumetric DDoS attacks. Furthermore, unlike the studies in [104], [105], mitigation modules in [106] will be applied by switches in an on-demand manner to optimize hardware resource limitation. At volumetric attack, the attacker sends a high amount of traffic or request packets to exhaust the bandwidth or resources of the victim. The detection logic identifies all attacks while mitigation modules are installed on demand to optimize hardware resource usage. The proposed method operates based on universal sketches which makes it possible to track a broad range of metrics with a single algorithm. The authors consider three components to implement mitigation: (i) filtering packets, (ii) analysis to identify malicious traffic, and (iii) update to the filtering. For each component, a library of mitigation functions is designed using switch-optimized logic. When attack postures change, a new resource allocation is computed using a near-optimal heuristic to redirect traffic to other available switches with the smallest rerouting cost. The in-network computations are universal sketch based attack detection and mitigation techniques that are implemented in Barefoot Tofino. The proposed method can mitigate attacks while keeping throughput at 380 Gbps.

4) Specific DDoS Attack Mitigation Cases: Duplicate address detection is essential in the configuration of IPv6 addresses, thereby all nodes in the same subnet will be able to join the network with unique IPv6 addresses and proceed with communication. In duplicate address detection, a node sends one or several Neighbor Solicitation (NS) messages, and it will configure an address when no Neighbor Advertisement (NA) messages from existing nodes are received. In a DoS attack there will be spoofed NA messages and IPv6 addresses can not be configured. The existent solutions to this attack, require either the modification of the protocol, or dependency on a central controller that suffers from single point of failure. To overcome these problems, Kuang et al. [111] propose a method that secures duplicate address detection in an in-network manner. The proposed method utilizes the programmable switches to filter forged NA messages without the need for a central control node or modification in the duplicate address detection protocol. The in-network computation, i.e., filtering forged NA messages is implemented by BMv2. Evaluation results show that the proposed method can prevent DoS attacks on duplicate address detection successfully with negligible overhead. Through in-network processing, the latency of NS/NA message processing has been reduced up to 40%.

Dimolianis et al. [112] assumes the use case of providing DDoS Protection services to universities and data centers. In the proposed mitigation method, P4 devices extract some features obtained from the traffic monitoring in the subnetworks. According to the features, anomaly detection is performed and appropriate alarms will be given to the mitigation systems. Three features are defined and a threshold based anomaly detection is advocated: (i) *The number and the dispersion of total incoming flows per epoch* which is calculated by adopting a moving average approach; (ii) *Subnet significance* which is calculated as the percentage of incoming flows to a specific subnet within duration of an epoch; (iii) *Packet symmetry* that is defined as the fraction of incoming to outgoing packets for a subnet during an epoch; the feature is used to avoid false detection of a subnet as a victim when it is the actual receiver of high volume of benign traffic. The aforementioned three features are compared by threshold values to decide about attack detection. In the implementation, P4 registers are utilized to implement required counters, arrays and probabilistic data structures. Furthermore, the analysis of the aforementioned features are included in the network element pipeline. The in-network computation, i.e., traffic feature analysis and attack detection is implemented in Netronome Agilio CX SmartNIC (Throughput 1-2 Mbps).

B. Firewall

Some studies have been carried out to propose in-network firewall solutions. Vörös and Kiss [115] provide a layer 3 firewall solution, Datta et al. [116] give a layer 3 and 4 firewall solution. Vörös and Kiss [115] implement a layer 3 firewall in a programmable router. In the proposed method, MAC and IP addresses that are violating security rules, or generating too significant packet rate are added to a list, namely called as *Ban List*. *Counters* in the router is utilized to measure metrics to define *Ban List*, e.g., the generated packet rate at each host, the number of attempts to establish connection. The proposed method extends the headers already defined to the router, i.e., Ethernet and IPv4 headers, by implementing IPv6 and UDP headers. After parsing TCP/UDP packets, fields of the packets are investigated not to be on a *Ban List*. The packets that match the *Ban List* will be dropped. In-network computation, i.e., layer 3 firewall policies are implemented in a P4 router.

Datta et al. [116] propose a configurable layer 3 and 4 firewall that is developed into software switches. The authors introduce a controller for the purpose of centralized management of the firewall. The high-level security policy is given as input to the controller, accordingly the controller sends the security rules to the switches. The security rules are implemented based on forwarding tables in the P4 data plane. Furthermore, controller communicates with the switches through a remote procedure call channel to activate or deactivate the firewalls. In-network computation, i.e., layer 3 and 4 firewall policies are implemented by BMv2. The proposed method is a co-design approach because the controller injects the security rules to the switches and have the centralized controlling over firewall policies.

C. Other Security Applications

Several research has been done in the scope of in-network security applications other than DDoS or firewall. A random forest based in-network attack mitigation has been proposed in [117]. A block chain based in-network attack detection has been proposed in [62]. The studies in [118], [119] focus on privacy threat for IP addresses. Network immunity against eavesdropping has been considered in [120]. Mitigation of network covert channels have been investigated in [121].

Laraba et al. [122] focus on mitigating Explicit Congestion Notification (ECN) protocol abuse. Security of Bring Your Own Device environment is the focus in [63], [64].

A random forest algorithm has been embedded in a programmable switch to detect attacks in the network [117]. The proposed method advocates in-network computing instead of transferring data to a central location to decide about the attack occurrence. Considering the memory limitation of the switches, they analyse UNSW-NB15 dataset [123] to select a subset of important features and select the number of decision trees to be implemented in the switches. Levels of the decision tree and the logic of decision is implemented in match-action stages. The parameters of the actions such as the threshold for comparison is configured by the control plane. The authors also provide mechanism for estimating some features like bit rate and TCP round-trip time which are used in learning process. Finally, they extend the idea to embed several trees used in the random forest algorithm in the switch. The in-network computation, i.e., random forest is implemented by BMv2. The proposed method detects more than 94% of attacks.

Yazdinejad et al. [62] propose a block chain based method to detect attacks in software defined networks. To enable block chain support, an architecture for packet parsing in the switch is defined which is able to extracts the blockchain header fields. Using the functions programmed by the SDN controller, the pattern of the arrived packets in the switches are investigated for attack detection. In the case that the attack is detected, a transaction will be carried out to perform the validation in the SDN control plane. The SDN controller checks the validity of the transaction and the validation result will be given to the switches. The in-network computation, i.e., attack detection is implemented in ZedBoard Zynq FPGA. The proposed method is a co-design approach since the validation step is performed by SDN controller. The results indicate detection rate above 70% for various attacks (e.g., DoS and Probe)

There exists a privacy threat for IP based Internet traffic at which information about communicating users and devices can be leaked by IP addresses. Existent approaches to obfuscate the IP addresses of the sender and receiver operate either by installing software on the user side (e.g., Tor browser) or applying some modifications to the network hardware. To overcome these problems, Datta et al. [118] propose an in-network surveillance mechanism. The proposed method encrypts IP addresses in packet headers before these packets enter the intermediate autonomous systems and decrypts source and destination IP addresses when the packets exit the intermediate autonomous systems. The proposed method also encrypts TCP sequence and acknowledgment numbers to prevent the adversary from recognizing which packets belong to the same TCP flow. The in-network computation is encryption and decryption of IP addresses.

Simlar to [118], Chang et al. [119] present an encryption based defense mechanism for IP addresses. The proposed method assumes that the sender/receiver switches are trusted while the switches located on the middle of the path can be malicious. The sender boundary switch performs a cryptography-based transformation on the address, and builds the corresponding data header. The other untrusted switches forward the data packet to the receiving switch based on the the flow table issued by SDN controller. At the receiving switch, the agreed parameters in the packet header are unfolded, the address field is decrypted, and the normal IP packet is given to the destined host. The in-network computation, i.e., encryption and decryption of IP addresses is implemented by BMv2 using the P4 language. Using the proposed method, the round trip-time of forwarding packets is less than 8 ms.

Liu et al. [120] present a method using programmable data planes to improve network immunity against eavesdropping. The proposed method consists of three defense lines. The first line consists of a forwarding policy at which the traffic packets are forwarded disorderly through various network paths and protocols (e.g., IPv4, IPv6). The second line provides disturb for eavesdroppers so that they can not classify the traffic appropriately. It works based on a transport encryption algorithm, at which traffic packets of a stream is distributed into multiple streams. In the third line of defense, the packets payload are encrypted by encryption-based countermeasures. The in-network computation are the aforementioned defense lines that are implemented with P4 in BMv2. Experimental results show that the proposed method can make eavesdropping difficult, and increase transmission throughput by 32% compared with baseline methods.

Xing et al. [121] have proposed a mitigation algorithm for network covert channels threats in cloud systems. They focus on two classes of network covert channel threats that are respectively called as timing channels and storage channels. As an example of the former, an attacker could use inter-packet delays to encode ones or zeros in a secret message. As an example of the latter, an attacker could embed secret data in the TCP sequence number or ACK fields. The authors offload as many primitives as possible to the data plane as a fast-path defense, and then perform a slow-path defense on the switch control plane for the rest, where there is powerful general purpose CPUs and RAMs.

The fast-path consists of three components: (i) connection monitoring which performs TCP monitoring and stores them as key-values; (ii) inter-packet delay characterization which approximates inter-packet delay distribution based on comparing the timestamp of the received packet and last-seen packet time stamp from the same flow; and (iii) storage channel defenses which adapts a set of defense techniques that manipulates some data inside packet to defense the attack. The slow-path defence has three more modules for channel defense: (i) statistical inter-packet delay tests which queries the inter-packet delay intervals for selected connections, and performs statistical tests for timing channel detection; (ii) timing channel defense which injects random delays to packets in suspicious connections to disrupt timing modulation; and (iii) performance boosters which can increase the performance of TCP connections to diminish the cost of performance reduction due to defence. The in-network computation, i.e., the fast path components are implemented on Tofino. The proposed mitigation algorithm is a co-design approach because some channel defense modules, i.e., the slow path, are implemented

on general purpose CPUs. In the experiments, in timing channel setup, the server is sending a file to the client, while in storage channel setup, the client is uploading a file to the server. The proposed method operates for the server defense. The proposed method achieves a data transfer time of 33 s for storage attack (0.3% increase in comparison with no defense). For timing channel the proposed defence method takes 60 s (3% increase in comparison with no defense).

ECN is a protocol that lies between IP and TCP layers and is used by the network switches to indicate possible congestion based on switch queue occupancy. A misbehaving TCP endhost can manipulate ECN bits, giving the sender the illusion that there is no possibility of congestion in the network. Laraba et al. [122] propose an in-network ECN abuse mitigation mechanism to operate within the network and without modifying TCP. First, the protocol specification is transformed into an Extended Finite State Machine (EFSM), and accordingly is extended with misbehavior states. Then, EFSM is converted to a program written in P4, and the compiled version will be installed on a PISA target switch. Finally, based on EFSM, the states of each connection are tracked and whenever a state labelled as misbehavior is entered, the predefined actions are triggered by the program, e.g., dropping the packet, rerouting the packet, generating an alert, applying corrective actions, etc. The in-network computation consists of tracking the flows based on the EFSM, ECN attack detection, and mitigation, that is implemented in BMv2. The evaluation shows that the proposed security approach can restore 25% throughput loss caused by misbehaving TCP end hosts.

Security in Bring Your Own Device environment where employees of the enterprise are allowed to use their private tablets, phones, and laptops at work is challenging. The security approaches implemented at server-side may not be efficient since they do not have access to context-information at the client side, furthermore they are not as fast as switches in processing. The studies in [63], [64] propose a context-aware in-network security solution. The authors propose a new language based on Pyretic Net-Core [114] to describe security policies. Blocking certain services in working hours, distance-based access control, and allowing access in the case that admin is online are samples of these policies. A module is installed at client side which is responsible for collecting device context information and embedding it in the network traffic. A compiler takes a policy program as input, and generates two outputs: (i) a configuration file for network elements, which describes the information that the client module should collect and embed in the packets; (ii) switch programs written in P4, which is to be deployed on the programmable switches to enforce the scurity policy. Finally, a runtime module is implemented in SDN controller that configures P4 programs on the switches, and communicates with network elements to deploy context configurations. This centralized controlling is only used for policy deployment or change, which is typically infrequent, while packet processing decisions are made directly on the switch. The in-network computation, i.e., security policies are implemented in BMv2 and Wedge 100BF Tofino. The proposed method can perform 1.2 million checks for a single security context, in a client-server based communication scenario.

D. Summary, Comparisons, Insights and Lessons Learned

In this section, we first briefly summarize the studies have been done in the scope of in-network security, and then discuss the insights and the lessons learned.

1) Summary: This section gives an overview of the research studies in the scope of in-network security. To mitigate DDoS attacks, the traffic is required to be rerouted to scrubbing servers on the cloud which imposes high latency, as well as cost for the operators in order to handle the attack. Offloading a fraction or whole functionalities required to detect and mitigate DDoS attacks in the network would reduce the latency and operational cost. Several studies have proposed in-network methods for DDoS mitigation. In-network flooding attack mitigation including SYN, TCP, and link-flooding have been studied in [60], [61], [98], [99], [100], [101], [102]. Generally, in flooding attacks, some detection primitives, e.g., cookie calculations, whitelisting estimation, statistics about SYN/ACK packets, statistics about TCP/UDP packets, entropy analysis of IP addresses, are offloaded to network elements. Furthermore, mitigation primitives, e.g., dropping packets, rerouting suspicious flows, are implemented as actions to be performed by the network element whenever the attack has been detected. AR-DDoS attack mitigation has been considered in [103]. Considering the security of the source of traffic, the study in [107] offload tag generation to the programmable routers for source address validation, while the studies in [108], [109], [110] filter spoofed traffic in the network element, e.g., by implementing *Hop Count Filtering*. A more general view of attack mitigation that includes several DDoS attacks have been presented in [104], [105], [106]. These studies offload collecting statistics about network traffic and connections, decision about attack occurrence (e.g., applying a threshold based method), or defence mechanism (e.g., dropping or following actions according to a specific control flow), to the network elements. Other specific in-network DDoS attack mitigation cases have also been investigated, e.g., duplicate address detection in [111], and offering DDoS Protection services to universities in [112].

There are several in-network firewall solutions: [115] at layer 3, [116] at layer 3 and 4. Generally, malicious packets will be detected at network element by rule-define mechanism implemented in TCAM, or through matching with a ban list. The packets will be dropped in the case that are recognized to be malicious.

Several in-network security studies have been carried out in the scopes other than DDoS or firewall. These include a random forest based attack mitigation [117], a block chain based attack detection [62], mitigating privacy threats for IP addresses [118], [119], network immunity against eavesdropping [120], mitigation of network covert channel threats [121], mitigation of ECN protocol abuse [122], and security of Bring Your Own Device environment [63], [64]. In these category of studies depending on the application, detection primitives, e.g., decision tree, TCP connection statistic calculations,

TABLE VIII
COMPARISON OF IN-NETWORK DDOS ATTACK MITIGATION. DS (DATA STRUCTURE), PLAT. (PLATFORM), Y (YES), N (NO), H (HADRWARE), S (SOFTWARE)

Scope	Ref.	Main Contribution	Methodology In-network Computation	Co-Design	DS	Evaluation Network Element	Plat.	Main Results
Flooding	[98]	Implementing SYN cookies and SYN authentication strategies using the P4 over off-the-shelf data planes	SYN cookies and SYN authentication strategies	N	Hash Table; Bloom Filter	T4P4S NFP-4000 Agilio SmartNIC NetFPGA SUME	H/S	Up to 13 Mpps throughput
	[61]	Proposing an in-network SYN flooding attack mitigation	Detecting SYN flooding attack based on the ratio of SYN/ACK and ACK/FIN packets; Dropping the packets from malicious source IP	N	Cache	BMv2	S	Up to 4000 bytes/s bandwidth saving
	[99]	Proposing a machine learning based attack detection mechanism with the focus on TCP flood attack	Traffic feature extraction	Y	-	BMv2	S	Detection accuracy over 98% 15 s Latency reduction
	[100]	Detecting and mitigating link-flooding attacks in the network through program implementation in programmable switches	Detecting and mitigating attacks	N	-	BMv2	S	Up to 80% throughput increment
	[60]	Implementing SYN and DNS anti-spoofing for DDoS attack using network elements.	SYN and DNS anti-spoofing functions based on primitives of SYN-cookie methods	N	Cache	Open vSwitch 2.3.1	S	Up to 278 $Kpps$ throughput
	[101]	Proposing a volumetric DDoS attack detection based on Shannon entropy analysis	Shannon entropy estimation; Calculation of characteristics of legitimate traffic; Threshold based attack detection	N	Count Sketch; Hash Table	BMv2	S	Accuracy 98% Latency 250 ms
	[102]	Proposing a fully in-network dynamic pushback mechanism to detect and mitigate volumetric DDoS attacks	Detect and mitigate volumetric DDoS attacks	N	Sketch	BMv2	S	Up to 94% accuracy Latency 0.1 ms
AR-DDoS	[103]	Proposing a distributed in-network detect and defense method against AR-DDoS attack	Detection and mitigation of AR-DDoS attack	N	Count-Min Sketch; Hash Table	BMv2	S	Accuracy 99.8%
Security of Source of the Traffic	[107]	Proposing a source address validation method based on packet tags	Tag generation	N	Hash Table	Commercial P4 switches	H	Up to 200 kbps bandwidth saving
	[108]	Providing P4-based implementation of several anti-spoofing mechanisms	Anti-spoofing mechanisms	N	-	NetFPGA SUME	H	Up to 7.5 Gbps throughput
	[110], [109]	Proposing an hop count filtering defense to filter spoofed IP traffic	IP2HC inspecting; TCP session monitoring; Calculating legitimate IP hit and spoofed IP hit statistics	Y	Hash Table	Barefoot Tofino	H	Bandwidth saving 200$Mbps$.
DDoS Mitigation in General	[104]	Proposing a method for DDoS attack detection and mitigation where employed edge programmable switches conduct analysis on the traffic	Statistic calculation about signature, SYN requests, TCP connection, and session establishment; Attack detection; Dropping malicious traffic	Y	Bloom Filter; Hash Table	BMv2	S	Up to 0.25 s detection latency
	[105]	Mitigation of DDoS attacks using in-network computing	Collecting statistic of packets; Defence operation	Y	Sketch; Hash Table	Barefoot Tofino	H	20 Gbps Bandwidth saving Tens of micro-second latency reduction
	[106]	Proposing a broad-spectrum detection and on-demand mitigation method for Volumetric DDoS attacks	Universal sketch based attack detection; Attack mitigation	N	Bloom Filter; Sketch	Barefoot Tofino	H	Throughput 380 Gbps
Specific DDoS Mitigation Cases	[111]	Proposing a method to secure duplicate address detection against DoS attack	Filter bogus NA messages	N	-	BMv2	S	Up to 40% latency reduction
	[112]	Proposing DDoS attack detection for the use case of National Research and Education Networks based on analysis of traffic features	Traffic feature analysis; Attack detection	N	Bloom Filter; Sketch	Netronome Agilio CX SmartNIC	H	Throughput 1-2 Mpps

security policies, as well as mitigation primitives, e.g., encryption/decryption of part of packets, and packet dropping, have been performed at network elements.

2) Comparisons, Insights and Lessons Learned: Tables VIII and IX compare the reviewed studies from the aspects of the contribution, methodology (in-network computation, co-design criterion, data structure of the network element used in method), and evaluation (network element, platform in simulation, main results). As it can be seen, the studies either fully implement the security mechanism in the

TABLE IX
COMPARISON OF IN-NETWORK FIREWALL AND OTHER SECURITY APPLICATIONS. DS (DATA STRUCTURE), PLAT. (PLATFORM), Y (YES), N (NO), H (HADRWARE), S (SOFTWARE)

Scope	Ref.	Main Contribution	Methodology In-network Computation	Co-Design	DS	Evaluation Network Element	Plat.	Main Results
Firewall	[115]	Implementing a layer 3 firewall in a programmable router using P4	Layer 3 firewall policies	N	-	P4 router	S	-
	[116]	Proposing a software-based layer 3 and 4 firewall that is incorporated into software switches	Layer 3 and 4 firewall policies	Y	-	BMv2	S	-
Other Security Applications	[117]	Embedding Random Forest Algorithm in programmable switches to detect attacks in the network	Random forest algorithm	N	Hash Table	BMv2	S	Accuracy more than 94%
	[62]	Proposing a blockchain-based method to detect attacks in software defined networks	Attack detection	Y	-	ZedBoard Zynq FPGA	H	Above 70% detection rate
	[118]	Proposing an encryption/decryption based surveillance protection method implemented in the network elements	Encryption and decryption of IP addresses	N	-	-	-	-
	[119]	Proposing a method to improve the security of SDN, through a defense mechanism based on encrypted IP address transformation.	Encryption and decryption of IP addresses	N	-	BMv2	S	Less than 8 ms latency
	[120]	Proposing a method with three defence-lines to improve network immunity against eavesdropping, which is implemented using programmable data planes	Multi path routing; Transport encryption algorithm; Packet payload encryption	N	-	BMv2	S	32% Throughput increment
	[121]	Providing a mitigation algorithm for network covert channels threats in cloud systems.	Fast path components	Y	-	Barefoot Tofino	H	Latency up to 60 s
	[122]	Proposing a detection and reaction method to Explicit Congestion Notification protocol abuse	Tracking the flows based on the EFSM; ECN attack detection; Attack Mitigation	N	Hash Table	BMv2	S	25% Bandwidth saving
	[63], [64]	Providing a context-aware security method for Bring Your Own Device environment	Security policies	N	Bloom filter; Hash Table	Wedge 100BF Tofino BMv2	H/S	Throughput as 1.2 million checks for a single security context.

network (the studies with the entry of 'N' for co-design) or follow a co-design approach (the studies with the entry of 'Y' for co-design) at which network elements are utilized in conjunction with servers or controllers in providing the required security mechanism.

Table X compares the fully in-network security schemes, server/controller based security, which is the baseline for comparison in many studies, e.g., [61], [99], [100], [105], and co-design schemes from the aspects of modeling state of system, detection model, attack detection accuracy, mitigation latency, and bandwidth consumption.

- *Modeling the State of System:* Fully-in-network security methods process and analyze the arrival traffic to the network element, thus have a local view of system. In contrast, in server/controller based schemes collected traffic generated from various nodes of the network is analyzed in a centralized manner thereby, there exists capability to have a global view of the system. Similarly, co-design schemes which can exploit general-purpose-computing can emulate having the global view of the system.
- *Detection Model:* There are hardware constraints (e.g., limited number of stages/pipelines/logical units, limited amount of match-action entries, registers, specific communications among logical units in FPGA) in implementing detection model in network elements. Thus, in comparison with server/controller-based schemes simpler models can be implemented in network element. In contrast, complex detection models like deep-neural-networks can be implemented in SDN controller or server to detect attacks [124]. In comparison with fully-in-network security schemes, Co-design schemes will have higher capability in implementing complex attack detection models due to the availability of general-purpose-computing unit. For example, in [99] the TCP/UDP packets feature extraction is offloaded to P4 switches, while the SVM and Random Forest learning models are implemented in conventional servers to analyze the extracted features and detect the TCP flood attack. This is in contrast with fully in-network schemes that mostly implement simple threshold based detection mechanism in network element, e.g., [101], [112], [117].
- *Attack Detection Accuracy:* The server/controller based schemes can achieve the highest accuracy due to capability of implementing complex accurate detection

TABLE X
COMPARISON OF FULLY IN-NETWORK IMPLEMENTED SECURITY SCHEMES, SERVER/CONTROLLER-BASED SECURITY SCHEMES, AND CO-DESIGN SCHEMES

Feature	Fully in-Network Security Schemes	Server/Controller-Based Security Schemes	Co-Design Security Schemes
Modeling State of System	Local view of system • Having statistics about the traffic passing through the network element	Global view of system • Having statistics about wide-network traffic	Can have global view of the system due to exploiting general purpose computation
Detection Model	Simpler than server/controller-based schemes	Capability to support complex detection model due to general purpose computation	Higher capability than fully in-network security schemes due to exploiting general purpose computation
Attack Detection Accuracy	Can be lower accuracy in comparison with server/controller-based schemes due to • Simpler detection model • Attack detection based on local view of system	Higher accuracy due to • Supporting advanced detection model • Attack detection based on global view of system	Can be higher accuracy in comparison with fully in-network security schemes due to exploiting general purpose computation
Mitigation Latency	Lower latency than server-based scheme due to • Processing of network elements with high throughput • Lower latency to detect attack due to omitting SDN controller or scrubbing servers • Performing mitigation as soon as the attack was detected	Higher latency	Can be lower than server/controller-based scheme • Some volume of traffic are processed at network element Higher than fully-in-network security scheme due to possible: • Processing of some traffic at controller/scrubbing server • Transmission of controlling signals from controller/scrubbing server to network element
Bandwidth Consumption	Lower • Transmission to scrubbing server/controller is not required	Higher	Higher than fully-in-network security scheme due to communication overhead between server/controller and network element

mechanism, as well as exploiting global modeling of system. The fully-in-network security schemes with the capability of simpler detection model and having local view of system might end to lower accuracy than server/controller-based schemes. The co-design schemes, can achieve higher accuracy than fully-in-network security schemes since they can implement more accurate complex attack detection models, as well as having the potential of global modeling of the system due to general-purpose computing utilization. As an example, the study in [99] which employs a general-purpose computation for TCP flooding attack detection module operating based on classification, besides an in-network feature capturing procedure, has reported the detection accuracy over 98%.

- *Mitigation Latency:* Fully-in-network schemes can have the lowest attack mitigation latency since network elements can process the packets at high throughput in comparison with conventional computing systems utilized in controller/server-based schemes. Furthermore, time will be saved as the attack can be detected without transmission of packets to SDN controller or scrubbing servers. Accordingly, mitigation can be triggered at data plane without SDN controller or scrubbing servers intervention. As an example, the proposed method in [111] can mitigate DoS attacks on duplicate address detection in programmable switches, with the latency reduction up to 40%. As co-design schemes, perform part of processing/decision at controller/scrubbing server (e.g., [109], [110]), and might need controlling signals/traffic be transferred between controller/scrubbing server and network element, they can experience more latency in comparison with fully in-network security schemes.
- *Bandwidth Consumption:* Fully in-network security methods which do not require traffic transmission to the scrubbing servers or (SDN) controller, will save bandwidth in comparison with controller/server based methods. For a source address validation application, the experimental results in [107] show that in-network filtering of the spoofed packets can save the bandwidth up to 200 kbps. The co-design approaches can consume higher bandwidth than fully-in-network security schemes due to overhead of communication between server/controller and network element. The transferring of extracted features in data plane to TCP flood detection module in [99] is an example of communication overhead between network element and server.

There are also some insights from which some lessons can be learned:

- *Diversity in Attack Detection:* One overall insight from our literature review of in-network security is that most of the studies provide security solution for DDoS attack, particularly flooding attacks. For many other attacks, few research has been done. For example, only one study considers AR-DDoS attack [103] or network covert channel threats [121], and only two studies provide firewall solutions, i.e., [115], [116]. However, there exist extensive types of attacks at which rerouting of traffic to remote servers to detect the attack will end to high latency and operational cost which can be unleashed through in-network security appliance. A learned lesson is that

more research effort is required to provide in-network security solutions for attacks other than flooding like AR-DDoS attack mitigation, firewall solutions, and network covert channel threats mitigation, etc.

- *Machine Learning Based Attack Detection:* Another insight is that most of the studies perform simple threshold based detection mechanism in the network element at which the collected statistics in the programmable network element are compared with threshold values to decide about the attack occurrence. Though the ALU requirement of the threshold based detection is simple enough to be implemented in many network elements with processing limitation, the detection accuracy however can be enhanced by applying more accurate detection mechanism. Machine learning techniques can gain higher accuracy. However, only [117] has used machine learning to detect attacks in the network. A lesson learned is that more research effort is required to apply machine learning in the network elements to detect attacks in the network.
- *Co-Design Security Schemes:* Another insight is that few studies, e.g., [104], [109], follow a co-design in-network security approach at which network elements are utilized in conjunction with servers or SDN controller in detecting and mitigating attacks. Co-design approaches can end to a more efficient attack detection. Though the programmable network elements are good candidate for estimating traffic statistic, each network element has a local view of traffic distribution that can not effectively determine the parameters to detect the attack occurrence. For example, to improve the detection accuracy, the parameters to detect the attack, e.g., threshold, can be defined by a server or SDN controller based on a global view of the traffic, while the traffic analysis can be performed in the network elements. We learn the lesson that more study can be done in area of co-design in-network security.

VI. In-Network Coordination

Participant entities in distributed systems, may require to agree on some data value or a sequence of operations that is needed for a computation or system operation. This agreement can be reached through execution of consensus protocols. The consensus mechanisms however, suffer high latency as multiple communication rounds among the participants are required to be completed to reach a consensus. Offloading parts/whole of functionalities required for the implementation of a consensus algorithm to the network elements will have the potential to reduce the latency. Besides consensus protocols, there are other coordination mechanisms in the literature that leverage in-network computing to speed up coordination. In this section, we provide an overview of the research done in the scope of in-network coordination. Fig. 2, Section VI illustrates the structure of this section.

A. Consensus Protocols

The Paxos consensus protocol participants, may play any of three roles: *proposers* who propose a value to the distributed system; *acceptors* who choose a single value; and *learners* who learn the chosen value. The protocol begins when proposers propose values, and ends when learners know the selected value by the acceptors. The whole protocol can be implemented through some iterations where at each iteration messages are transfered among the roles which can be deployed in the servers.

Paxos can be performed in two phases. In the first Phase, a proposer selects a round number and sends a prepare request to a portion of acceptors. When a prepare request with a round number larger than previously received round numbers is received, through replies the acceptor promises the rejection of future requests with smaller round numbers. However the accepted value and corespondent round number will be returned to the proposer in the case that the acceptor already has accepted a request. When the proposer receives replies from a portion of acceptors, the second phase starts.

The second Phase provides a procedure through which a value would be selected by the proposer. The proposer will select a new value when it receives no value in the replies. On the other hand, in the case that some values have been received in the first phase, the proposer will select the value with the highest round number. After the selection, the proposer will send an accept request including the selected value and the associated round number to the same fraction of acceptors. Accordingly, the acceptors will acknowledge the receipt and send the accepted value to the learners, unless the acceptors have already acknowledged another request with a higher round number. When a fraction of acceptors accepts a value consensus will occur.

The involved roles in the consensus protocol, can be implemented in network switches in order to reduce the message traverse path in the network, thereby reducing the latency to reach consensus. Using P4, Dang et al. propose an in-network computation for the second phase of the Paxos protocol in [125], [126]. In [127], the authors describes a complete Paxos implementation including both phases in the network. The in-network computations have been implemented in Tofino and NetFPGA SUME. The studies in [125], [126] that offload a fraction of consensus protocol to the network are co-design approaches, however, the study in [127] that implements the consensus protocol fully in-network is not considered as a co-design approach. They present an open-source implementation of Paxos with at least 3× latency improvement and 4 orders of magnitude throughput improvement in comparison with host based consensus in data centers.

The authors in [128] define an Ordered Unreliable Multicast (OUM) communication in data centers based on which they define NOPaxos as a replication protocol. The authors assume a tree structure at which there are connections between top-of-rack switches and aggregation switches. In a higher level above aggregation switches the communication is provided by core switches. In OUM communication, a client sends messages to a group of recipients. There is no guarantee for message delivery, however there are guarantees in the order of received messaged in recipients. To reach this aim, all packets with destination of a specific OUM group, will be sent through a

single sequencer, for the purpose of inserting a sequence number to each packet prior to routing toward destination. The forwarding rules will be organized by SDN controller to route messages toward the sequencer and the group members. The authors discuss three possible implementations for sequencer: programmable network switch, network processor, and end host. For the purpose of fault tolerance, if the sequencer fails or be disconnected from OUM group members, a new sequencer will be selected and configured by the controller. Based on OMU communication, the authors propose the replication protocol NOPaxos based on a coordination of a leader, at which the replicas agree on the sequence of requests to be executed. The in-network computation, i.e., sequencer is implemented on Cavium Octeon II CN68XX network processor. As sequencing is a fraction of replication protocol, the proposed method is a co-design approach. The evaluation results show that latency is below 200 μs and the throughput is above 50 *Kops/s*.

Raft is a consensus algorithm that abstracts a replicated log. The clients send their requests to an entity namely called the leader. Leader stores the requests in a log to guarantee a total order, and accordingly, it replicates the requests to the followers through an *append entries* request. The followers append the request to their logs and notify the leader in order to execute the operation and respond to the client. To improve latency, Zhang et al. [129] offload the processing of *append entries* messages to the P4 switches. As the leader and the follower functionalities that are required to complete the consensus, are implemented in non-network elements, i.e., container, the proposed method is a co-design approach. According to the evaluations, the latency between a leader and follower is roughly less than 937 μs.

Kogias and Bugnion in [130] propose a kind of consensus protocol namely called HovercRaft, based on Raft, in order to improve the performance of state-machine replication for microsecond-scale data center services. The authors identify the bottlenecks that might arise in the consensus communication pattern. Replication of the request to the followers, replying to all clients, and packet processing of the majority of followers are the IO and CPU bottlenecks for the leader. To overcome these bottlenecks, the authors suggest some modifications in the consensus communication pattern. Theses modifications include: Separating replication from ordering, and replicating the requests to all nodes through IP multicast; balancing the load of replying to the clients between the leader and the followers; Finally, implementing part of the leader functionality, i.e., aggregator to handle the *append entries* requests and the replies, in the network. The in-network computation, i.e., aggregator is implemented in Barefoot Tofino. As aggregation is a fraction of consensus protocol, the proposed method is a co-design approach. For delivering up to 1 million operations/s for clusters of up to 9 nodes, results illustrates a 4 $\times$ speedup for the YCSBE-E benchmark running on Redis over an unreplicated deployment.

Distributed architecture for software defined networks includes multiple controllers where each controller replicates the states of other controllers. However, to make the same decision for a request, the states are required to be consistent through a consensus mechanism. Though leader-based consensus methods (e.g., PAXOS, RAFT) can encounter with node failures they can not cope with Byzantine failures such as software bugs, and malicious attacks at the controllers. This can end to incorrect replication states in the controllers and can bring unintended operations, e.g., malicious modification of the message. To keep the correct operation in a distributed SDN through Byzantine Fault Tolerance (BFT), the efficiency and scalability would be compromised due the delay and traffic load imposed by the consensus procedure. To overcome this overhead, Sakic et al. in [131], [132], and Han et al. [133] leverage in-network computing.

Sakic et al. [131], [132] focus on providing a correct consensus in the scenario of Byzantine failures where a subset of controllers operate faulty. Three entities are involved in the model: (i) *Network controllers* that decide about forwarding plane configuration. The corrupted operation or malicious controller however, may diverge from the decision of correct controllers. (ii) *P4-enabled switches* that do the operation of packet forwarding for both controller commands and applications. Whenever a switch is defined to be the *processing node*, it would distinguish the correct configuration messages generated by different controllers; (iii) *Reassigner* that performs dynamic assigning of the switches to controllers for the purpose of configuration based on the the operations of the controllers. Furthermore, it determines the switches playing the role of processing node. The authors propose an optimization formulation to select the optimal processing node. The in-network computation, i.e., distinguishing the benign controllers, is implemented in BMv2 and Netronome Agilio SmartNIC. The proposed method is a co-design approach as the processing node is selected through an optimization technique implemented using general purpose CPU. Due to filtering of incorrect messages, the control plane traffic load has been reduced by 33% and 40% on average, in respectively 128 random switches and Fat-Tree topologies.

In the context of the same SDN architecure as in [131], [132], Han et al. [133] offload complete functionalities of the BFT to the programmable switches. Furthermore, time synchronization and state synchronization are also performed in programmable switch to reduce the communication and latency overhead imposed by communication among controllers. The aforementioned in-network computation is implemented in BMv2. Simulation results show that proposed method ends to 80% reduced response time compared to conventional BFT consensus mechanisms.

B. Other Coordination Applications

Beyond consensus protocols, there exists other in-network coordination applications in the literature. A lock management system has been presented in [134]. A coordination mechanism based on group cast communication for transaction processing system has been considered in [135]. A coordination mechanism based on in-network consensus to maintain consistency in data centers has been proposed in [136].

Yu et al. [134] utilize in-network computing and propose a lock management system. Requests of clients are sent to the locking system which processes the requests with a coordination of switch and servers. In more details, management of locks are distributed among the lock servers. Getting information from a directory service, the destination IP at each request will be set to the server IP responsible for the required lock. Upon a request arrival at a switch, the switch will grant the lock for that request if it holds the information about the requested lock object, and the lock is available. However, in the case that the lock is not available, the request will be inserted into a queue if there is sufficient memory. In the case that the switch does not hold the lock object information or the memory is not adequate, the request will be forwarded to the lock server defined by the destination IP. The authors also discuss the possibility of lock and data acquisition in the same round time to reduce round trip time. From the aspect of implementation, register arrays are utilized to queue the requests for the locks, and a specific UDP port is defined for the locking service. A lock request packet contains several fields including action type (lock acquire/release), lock ID, transaction ID, and client IP. Furthermore, the mapping of lock ID to the associated register array, as well as operations to grant and release locks are implemented through match-action tables. Finally, the authors formulate the problem of allocating locks to the switches as an optimization which is similar to fractional knapsack problem and is solved in polynomial time. The in-network computation, i.e., locking management is implemented in Barefoot Tofino. The proposed approach is a co-design approach as lock servers in conjunction with switches are handling locks due to memory limitation in switches. Proposed method improves the throughput by 18 $\times$, and reduces the latency by 20 $\times$ over baseline solutions.

A transaction processing systems at which clients perform transactions over a distributed storage system structured as a combination of shards has been considered by Li et al. [135]. The authors utilize in-network computing to coordinate the transaction with higher performance. By manipulating IP and UDP headers, the proposed protocol introduces a groupcast communication where a message is transmitted to several multicast groups with ordering guarantees. Groupcast communication is implemented using a centralized sequencer which can be replaced by SDN controller whenever it fails. The sequencer itself can be implemented in several ways including in-switch device, network middleware, and end host which the authors believe the highest performance is achieved by implementation in a switch. The whole protocol is divided into three layers: (i) The in-network concurrency control layer that operates within and across shards, and provides a consistent ordering of transactions without guaranteeing the reliability in message delivery; (ii) The independent transaction layer which is responsible for the reliability and atomicity of operations; (iii) The general transaction layer that provides isolation of transactions, by constructing the transactions using independent elements. The in-network communication, i.e., sequencer is implemented with the P4 language, in Cavium Octeon CN6880 network processor. The approach is co-design as a fraction of required functionalities is implemented in the network. The evaluation results achieves up to 35 $\times$ higher throughput and up to 80% lower latency than a conventional design on standard benchmarks.

C. Summary, Comparisons, Insights and Lessons Learned

In this section, we first briefly summarize the studies have been done in the scope of in-network coordination, and then discuss the insights and the lessons learned.

1) Summary: This section gives an overview of the studies carried out in the scope of in-network coordination. Several consensus protocols have been proposed in the literature. These protocols facilitate the participants to reach consensus, i.e., agree on some data value or a sequence of operations required for system operation. Offloading parts/whole of functionalities required for the implementation of a consensus algorithm to the network can reduce the consensus latency. In this line of research, second phase of the Paxos protocol [125], [126] has been implemented in the network element, while a full implementation of this protocol has been presented in [127]. In an ordered unreliable multicast communication proposed in [128], module of sequencer with the role of adding a sequence number to each packet before forwarding it to its destination, has been implemented using the programmable network switch. A fraction of Raft consensus protocol in [129], and a modified version of it in [130] has been offloaded to programmable switches. Byzantine Fault Tolerance functions in the proposed protocols for persistence and correct operations in distributed SDN have been offloaded to programmable switches [131], [132], [133].

There are also other coordination mechanisms in the literature that leverage in-network computing to speed up coordination. A lock management system has been presented in [134]. A coordination mechanism based on group cast communication for transaction processing system in [135] perform sequencing in programmable switch. Finally, a coordination mechanism for the purpose of consistency in data centers in [136] leverage in-network computing for the purpose of broadcasting.

2) Comparisons, Insights and Lessons Learned: Table XI compares the reviewed studies from the aspects of the contribution, methodology (in-network computation, co-design criterion) and evaluation (network element, platform in simulation, main results). An overall insight is that most in-network consensus studies offload a part of functionalities of the proposed protocol to the data plane. Full implementation of consensus protocols in data plane have the potential to yield more efficient performance than a partial offloading scenario as shown in [127]. A lesson we learn is that more research effort is required to provide consensus protocols that are fully implemented in data plane.

Another insight is that only three studies, i.e., [134], [135], [136] have been presented that perform coordination other than consensus. A learned lesson is that more research effort is required to leverage in-network computing to provide coordination solutions other than consensus protocols, for network and distributed systems.

TABLE XI
COMPARISON OF IN-NETWORK COORDINATION STUDIES

Scope	Ref.	Main Contribution	Methodology In-network Computation	Co-Design	Evaluation Network Element	Plat.	Main Results
Consensus	[125], [126], [127]	Proposing Paxos consensus as a network services	Paxos consensus protocol	Y([125], [126]), N([127])	Tofino NetFPGA SUME	H	More than 3× latency reduction More than 4× throughput increment
	[128]	Defining the ordered unreliable multicast communication protocol for data center networks, and accordingly introducing a replication protocol	Sequencer	Y	Cavium Octeon II CN68XX network processor	H	Latency below 200 μs Throughput above 50 $Kops/s$
	[129]	Proposing an in-network implementation of Raft consensus protocol	Processing of *append entries* messages	Y	P4 switch	S	Latency below 937 μs
	[130]	Proposing a consensus protocol for the purpose of the resilience and the performance of general-purpose state-machine replication for microsecond-scale data center services.	Aggregator	Y	Barefoot Tofino	H	4× Latency reduction
	[131], [132]	Proposing a method for a SDN architectur, to enable correct consensus in scenarios where a subset of controllers is faulty	Distinguishing the benign controllers	Y	BMv2 Netronome Agilio SmartNIC	H/S	40% Bandwidth saving
	[133]	Proposing a switch-centric Byzantine Fault Tolerance mechanism in distributed software defined networks	Full set of BFT functions; Time and state synchronization	N	BMv2	S	80% Latency reduction
Others	[134]	Proposing a lock management system using a combination of top of rack switch and multiple lock servers	Locking management	Y	Barefoot Tofino	H	By 18× throughput increment By 20× latency reduction
	[135]	Proposing a coordination mechanism based on group cast communication for transaction processing system	Sequencer in group-cast communication	Y	Cavium Octeon CN6880 network processor	H	Up to 35× throughput increment Up to 80% latency reduction

VII. TECHNOLOGY SPECIFIC IN-NETWORK COMPUTING APPLICATIONS

In this section we study technology-specific in-network computing applications in cloud computing, edge computing, 4G/5G/6G, and network function virtualization. Fig. 2, Section VII illustrates the structure of this section.

A. Cloud Computing

The studies in the scope of cloud computing, either provide load balancing for data centers, or resource allocation solutions.

1) Load Balancing: The packets destined to a data center service with a virtual IP address (VIP) can be mapped with a pool of servers (DIP pool). In-network computing can eliminate the need for a software based load balancer, in consequence the high cost of servers for load balancing, as well as high latency and jitter in software implementation will be diminished. Miao et al. [137] have leveraged in-network computing and provided a load balancer in data center through implementing two tables namely called *connTable* and *VIPTable* in the switches. *VIPTable* defines the pool of DIPs that can be mapped to VIPs. On the other hand, *connTable* maps every TCP connection to a DIP, i.e., the connection packets will go through the associated DIP to receive the required service. Two major challenges have been considered: (i) To store large number of connections in *ConnTable*, with limeted SRAM, a hash digest of a connection is stored instead of the actual connection key. Furthermore, a DIP pool version is stored instead of the actual DIPs; (ii) The authors describe a phenomena at which the DIP pool might be updated within life time of a connection by adding or removing servers to the pool. They define per-connection consistency as a requirement that all packets of a specific connection be associated to the same DIP within the lifetime of a connection. A bloom filter is used to track the connection arrival and accordingly, a consistent DIP mapping mechanism for the connections is implemented. The in-network computing, i.e., load balancing is implemented in the P4 switch. The proposed method can balance the load of ten million connections at line rate.

Ye et al. [138] propose a multi-path load-balancing method in data centers. They consider a tree-like connection structure (with possibility of multiple roots) among P4 switches, at which every switch have a table to keep the utilization of all paths to the leaf switches. In the proposed method, each switch has knowledge about link bandwidth of its ports, thereby the utilization can be estimated as the ratio of the transmission rate to the bandwidth. According to the proposed method, the traffic is handled at the granularity of flowlets. When a packet

arrives to a switch, the inter-arrival time is calculated by comparing the current and previous timestamps. If it is greater than a threshold, the packet is considered as a new flowlet. Then, weighted utilization table is used to select a new path for the flowlet. The in-network computing composed of detecting flowlets, estimating utilization of paths, and packet routing which are implemented in BMv2. For a topology of leaf, spine, core and for client-server based requests, the proposed method has reduced flow completion time up to 2%.

A transport protocol for latency-sensitive Remote Procedure Call (RPC) calls in a datacenter has been presented in [139]. The proposed protocol is a request/reply-oriented protocol without saving the state across requests, at which the request identified by some parameters (e.g., source IP, UDP port, and an RPC sequence number) is initiated by the client. The request destination is decoupled from the server that will process the request, thus relaxing the semantic of point-to-point RPC communication. The router identifies a suitable target server according to load balancing policy and directs the message to it. In order to reduce the router processing bottlenecks, the router only decides about the first packet of a request, while the remaining is transmitted to the same server. The servers send feedback messages to the router about their status of idleness and availability that are used on the load balancing decision. The authors have implemented a Random, Round-Robin, JSQ and join-bounded-shortest-queue (JBSQ) load balancing policies on the software router whilst implementing only JBSQ for Tofino data plane due to more simplicity. In comparison to a baseline method, the latency of Web index searching on a cluster of 16 workers has been improved by 5.7 $\times$. In comparison with a baseline method, the throughput of the key-value store requests on a 4-node cluster with master/slave replication has been improved by more than 4.8 $\times$.

In contrast to [137], [138], [139] which advocate a full implementation of load balancing at network elements, Gandhi et al. [140] leverages a combination of in-network computing and software implementation to provide a more flexible load balancing for datacenters. There are two limitations for software-based load balancers: limitation in the capacity of processing packets; and high variable latency. To cope with these limitations, the authors use programmable switch that already exists in the data centers to deploy a scalable, high performance load balancer. They exploit unused entries in ECMP and tunneling tables in a switch to perform traffic splitting and packet encapsulation. Indeed, a database that maps virtual IP address to the direct IP, is stored in a switch to be able to balance the load. As load balancing functionalities, i.e., traffic distribution and encapsulation are managed in the data plane, it can have low latency/cost, while having high capacity. On the other hand, management of switch failures is challenging, thus there will be a compromise for flexibility in comparison with a software based load balancer. To overcome this limitation, the authors utilize a combination of the switch and a software load balancer. The traffic mainly is managed by the switch, while software load balancer performs the role of a backup, to enhance flexibility. The authors also propose a greedy algorithm to decide about partitioning of mappings among switches to overcome the memory limitation of switches. The in-network computing, i.e., traffic splitting and packet encapsulation is implemented in simulation based switch. The proposed method is a co-design approach since it leverages a combination of software-implemented load balancer and programmable switches to perform load balancing. The results show that the proposed method provides 10 $\times$ more capacity than the pure software load balancer, at a fraction of the software-load balancer cost, while also reducing the latency by 10 $\times$ or more.

2) Cloud Empowered With INC: Resource Allocation Studies: This category of studies provide cloud resources allocation solutions in cloud computing environment that is augmented with in-network computing. An online resource allocation has been presented by Tokusashi et al. [27]. The authors develop a scheme to dynamically offload computing of an application between servers in data centers and the network. To decide about in-network computing, two types of controllers are proposed: *network-controlled* and *host-controlled*. In *network-controlled*, when the mean number of messages that are exchanged by the application deviates a predefined threshold, the workload is processed by the network. In *host-controlled*, when the application power consumption or CPU usage exceeds a threshold, the controller offloads the workload to the network. The in-network computation, i.e., the application execution is implemented by NETFPGA and Tofino. The proposed method is a co-design approach since it utilizes a combination of data center servers and network elements to serve the applications. The experiments show that the power consumption of a software system on commodity CPU can be improved by 100 $\times$ using an FPGA, and 1000 $\times$ using ASIC implementation.

Blöcher et al. [28] propose resource allocation in a hybrid of data center servers and network resources environment. Tenant describe its application with some predefined APIs and submit it as a directed graph of composites. The collection of composites is defined through some templates in a composite-store repository. Each composite can be a functionality like aggregation, load balancing, caching, data base functionality, each can be carried out by a combination of in-network computing nodes and data center servers. The tenant also defines the characteristic of the in-network computing nodes or servers it requires, like the number of CPU cores and RAM size for servers and the programming version and the throughput for the in-network computing nodes. Once the application is submitted, it is transformed into a polymorphic resource request which is a graph of server and network task groups. The resource allocation for the polymorphic resource requests is modeled as optimization with the objective of allocating resources to maximum number of requests while respecting server and in-network computing resource constraints. Heuristic is proposed to solve the optimization. The proposed method is a co-design approach since it utilizes a combination of data center servers and network elements to serve the applications. Experiments with a workload trace of a 4000 machine cluster shows reducing network detours by 20%, and placement latency by 50%.

Wu and Madhyastha [141] consider a situation where a third-party namely add-on providers can augment cloud provider capabilities by exposing APIs to new services. In such multi-provider environment, P4 programs can be installed in network switches to decide about forwarding packets to appropriate destination that can be an add-on VM or a cloud service, according to a recognition process that being handled by add-ons is required or not. The in-network computing is distributing packets between add-on VMs and cloud services.

B. Edge Computing

This section gives an overview of INC applications specifically for the technology of edge computing. First, we provide overview of the studies that apply in-network computing in the favour of edge intelligence provisioning. Second, we give an overview of the studies that discuss resource allocation/task offloading methods in an edge computing empowered with in-network computing. We have seen task offloading methods as a kind of resource allocation since in task offloading somehow the decision about allocation of resources to tasks are performed as well. Third, we will discuss about security mechanisms we found about in-network computing application in the scope of security in edge computing.

1) Edge Intelligence: In this category of research, [15] exploits in-network computing to perform critical subtasks involved in provisioning of edge intelligence service for mobile edge computing to speed up the whole intelligence/controlling scenario. On the other hand, the studies in [69], [81] perform optimizations on federated learning procedure through in-network computing. Mai et. al. in [15] propose a mobile edge architecture enriched with in-network computing for industrial IoT, where critical subtasks with low latency requirement that are involved in providing intelligence and control at edge, are offloaded to the network elements, while the rest of subtasks are performed by the mobile edge computing. In the proposed architecture, the connectivity between sensors/actuators and MEC server are provided through programmable network elements. Two use cases are considered: robotic motion control and fire detection.

For the use case of robotic motion control, the action is defined by a motion control mechanism as a multiplication of two vectors *current robotic state vector*, and *parameter matrix*, at which the *parameter matrix* is learned through machine learning. The control matrix multiplication which needs less computing and storage capacity, is offloaded to the programmable switch, while implementing the more complex process of learning of parameter matrix in the MEC. The state data collected by the robotic sensors, are encapsulated in UDP datagrams and sent toward the network element which parses the incoming packet headers in order to extract the state data. Then, the multiplication operation is performed and the result will be returned to the robot. In the same time, the UDP packets will be forwarded to the MEC to perform the machine learning operation in order to update the *parameter matrix*, and accordingly the matrix will be updated at the network elements.

For the fire detection use case, a Complext Event Processing (CEP) engine is utilized to detect the potential fire danger according to the streams of monitored data at sensors. Through a tool, the application function is converted into a set of "match-action" that analyzes the data stream for the purpose of fire detection. The matching operations will be performed at the programmable switch, while the compute and storage-intensive process of rules learning are performed in the MEC. Whenever a complex event, i.e., fire danger signal, is detected, the switch will send the alarm information. Whenever the CEP engine receives several consecutive events of temperature above a predefined threshold value, a matching will occur. The in-network computation, i.e., lightweight critical subtasks (e.g., matrix multiplication in a robotic motion controlling application) is implemented in a P4 switch. The proposed method is a co-design approach since it utilizes MEC servers for complex tasks like learning. The evaluation results show that the complex events can be successfully detected.

Qin et al. [81] proposes a federated learning for a system consists of a central cloud and several edge network domains. For each domain, there is a gateway node responsible for: (i) forwarding packets from/to the devices of that domain, (ii) a neural network-based binary classification. The gateway extracts bits from incoming packet's header and gives them as input to a neural network. While a neural network classification is performed at the gateway, aggregation after local updating is performed at the cloud. The in-network computation is the neural network which is implemented using BMv2. The proposed method is a co-design approach due to cloud involvement in aggregation. For a use case of malware traffic detection, the evaluation shows that federated learning has enhanced detection accuracy roughly 20% (accuracy of roughly 95%) in comparison with the case without federated learning.

Advances on service-centric networking, software-defined networking, as well as edge intelligence have impacted future networks. Li et. al [69] jointly considers these techniques to present the concept of in-network intelligence. The authors explain a content/computation/context-aware adopted version of service-centric networking protocol at which the nodes in a software-defined network can announce their interests for content and/or computation for a specific context and be served by nodes capable to provide the content/computation for the requested context. The proposed protocol enables in-network computing among edge nodes. The authors focus on a federated learning as an application of edge intelligent at which an aggregator and several workers at edge of network perform federated learning. The authors define a scenario at which the workers for performing local training on a specific data can send their interest to other nodes for the content of the data, and those nodes will directly perform local training via in-network computing in the case that they have the data. This offloading of training through in-network computing reduces the communication overhead. The in-network computation are caching and model training which are implemented in Open vSwitch. However, the evaluation is limited. For a small topology of network with four switches and one user, for a federated learning in the application of image classification based on convolutional neural network, the authors have

reported only the traffic rate in the network and do not provide any comparison for learning.

2) Edge Empowered With INC (Resource Allocation/Task Offloading Studies): This category of studies provide resources allocation or task offloading solutions in an edge computing environment that is augmented with in-network computing. While [58], [142], [143] tackle the resource allocation problem with an optimization approach, the study in [144] provides an architecture with the capability of task offloading to an edge processor enhanced with an FPGA accelerator.

Ali et al. [58] focus on use cases like open-air rock concerts and sports events at which constructing the wired network is not beneficial. The authors, design and implement an architecture for the required services over a Wireless Mesh Network, at which microservices are implemented in semi-permanent wireless mesh devices. Indeed, mesh network elements, e.g., mesh relay nodes, mesh routers, etc., that primarily route the packets, serve as fog nodes for in-network computing in order to provide the microservices. The proposed architecture includes three layers of IoT, fog, and application, as well as a fog controller. The devices in the layer of IoT, generate data, while APIs are provided at the fog layer in order to place microservices in wireless fog nodes for the purpose of computation. Furthermore, the network information including memory/storage/CPU-utilization of fog nodes, as well as the information about link loads, and packet drops are used to decide about the fog node that should perform the computation. A repository consisting of already-built containers are utilized in the application layer, to provide on-demand containers to install microservices in the fog nodes. Communication among fog nodes, IoT devices, and microservices are controlled by fog controller. Furthermore, selection of fog nodes for microservice hosting are performed by fog controller through an algorithm that minimizes response time. The in-network computation, i.e., microservice execution is implemented on mesh network elements. For a scenario of end-to-end message delivery the latency of the proposed method is less than 6 ms for a fog network of less than 650 nodes.

Lia et al. [142] presents a resource allocation solution for an edge computing domain at which SDN-enabled nodes are utilized to execute the given input tasks. The placement of computing tasks at the SDN-enabled nodes is modeled as an Integer Linear Programming problem with the objective of network usage minimization while respecting the predefined tasks delay constraints. The optimal solution has been found by CPLEX optimization tool. The in-network computation is task execution which is performed by SDN-enabled nodes. The simulation results illustrate that with delay constraints up to 100 ms, the task offloading to cloud has been reduced by a factor of 99%, while the amount of exchanged data has been reduced by a factor of 10 $\times$ in comparison with a cloud-based computation approach.

Cooke and Fahmy [143] considers a distributed and hybrid computational environment consisting of computing nodes distributed in various layers including from layer at very edge of the network, middle layers (e.g., gateways, and routers), as well as cloud data center. The nodes have capability of both software and hardware-accelerated implementation. The problem of deciding about implementation of computation nodes, as well as allocating nodes to tasks has been modeled as an optimization problem. With a focus on an object tracking use case (by camera), they have evaluated the effect of distributing tasks with various implementation scenarios on performance metrics including latency, throughput, energy consumption, as well as cost. Various scenarios apply in-network computing on various layers. The in-network computation is task execution which is implemented by FPGA acceleration. The proposed approach is co-design since tasks are executed by both software (on general-purpose computation nodes) and hardware-accelerated resources. For the aforementioned use case, adding FPGA accelerators has gained latency of 0.8 s, throughput of 133 camera frames per second, and energy of 1.56 (unit of energy), while implementing with cloud processing has gained 1.9 s, 3.4 camera frames per second, and 30 respectively for latency, throughput and energy.

Xu et al. [144] propose a FPGA-based accelerated method deployed at the network edge for the purpose of offloading compute-intensive tasks to accelerate the mobile applications. The authors propose a system architecture at which mobile devices are connected to a wireless edge network including a WiFi router and an FPGA board connected via Ethernet. Edge offload manager component within wireless edge network would decide either to carry out the requested computation within an edge processor enhanced with an FPGA accelerator or forward the request to a remote cloud (See Fig. 8). Three applications has been considered: (i) Handwritten digit number recognition which operates based on a convolution neural network model similar to LeNet-5. (ii) Object recognition which is performed based on a binarized neural network model. (iii) Face detection which operates based on a similarity-checking based computer vision algorithm. The in-network computation is neural network inference and computer vision algorithm operations which are implemented in ARMA-FPGA board (Xilinx Zc706). The proposed method is a co-design approach since only the operations that can be performed in accelerator are offloaded to FPGA, while the rest are performed in a general-purpose CPU. Experimental results shows that compared with general purpose CPU-based edge/cloud offloading, the proposed method reduces the response time and execution time by up to 3 $\times$ and 15 $\times$ respectively and saves 29.5% and 16.2% of energy for mobile device and edge nodes respectively.

3) Security: In this section, we explain security mechanisms we found among our surveyed studies in the scope of edge computing. Migrating access control procedures running on centralized servers to the edge data plane equipment, in order to reduce the transmission overhead and improve service capacity, in massive machine communication for industrial IoT, has been proposed by Song et al. [145]. Fig. 9 illustrates the structure of proposed method. Various heterogeneous terminals send requests for different services, e.g., sensing, data upload, and required control signaling. The whole procedure is done in four steps: (i) System initialization: at this step, administrator sets up appropriate access control policies through a network management application within the control plane. The

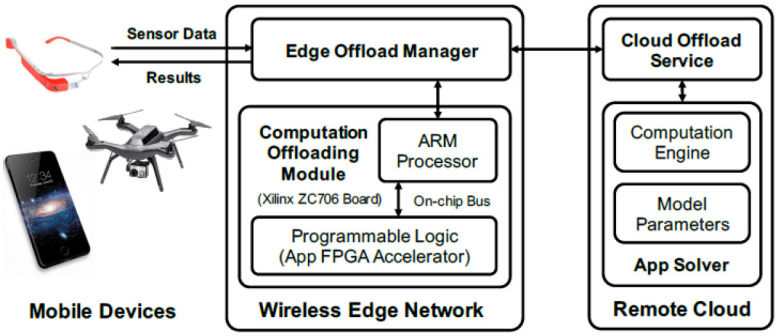

Fig. 8. Computation at edge enhanced with FPGA accelarator [144].

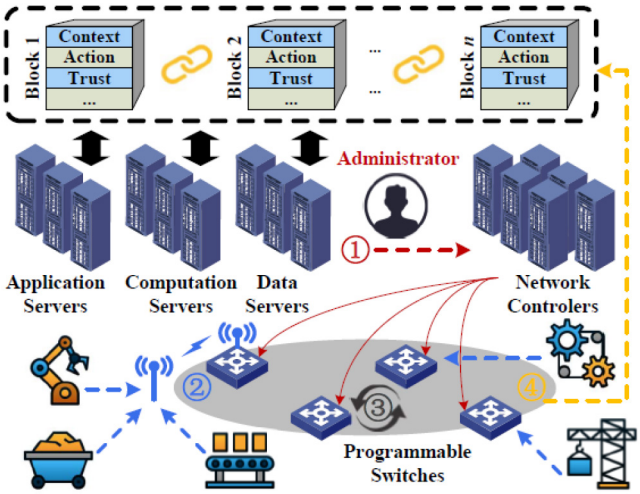

Fig. 9. The structure of the access control for massive machine communication [145].

policies are defined according to some context variables, e.g., the sensing frequency of the terminal, the distance between terminal and service object. The actions are defined based on the values of the context variables, and the whole policies will be set up by controller in edge programmable switches close to terminals. (ii) Request sending: A terminal sends its request for a service as well as the context information to the access equipment. (iii) Request process: The access equipment, i.e., switch extracts the context, from packet header, and according to the matched service object applies the corespondent rule, i.e., allow, deny, alert, re-authentication. There will be real-time interaction with the controller for updating the polices. (iv) Secure logs generation: The access behaviour of terminals will be recorded and assessed, in order to calculate a trust value for the terminal based on reported contexts, request upload characteristics, and the history of terminal's misbehavior. The switch will calculate the statistics, e.g., size, frequency, and requested service objects. Base on the logs, sophisticated attack detection (e.g., DoS attack) methods can be implemented on server. To ensure the trustworthiness of logs, the authors employ blockchain as the data structure. To cope with switch storage limitation, an algorithm is proposed that dynamically deploys the policies on the switches. The in-network computation is calculating the statistics of accessing services, as well as running access control policies which are implemented in BMv2. The proposed method is a co-design scheme due to real-time interaction of controller with data plane for updating policies as well as attack detection which is implemented in server. In comparison with a centralized approach, the authorization decision time, has been reduced up to 8 ms due to pipeline speed of switch as well as processing at edge near the terminals. Through filtering of illegal traffic, as well as responding at edge, the transmitted traffic volume in the network has been reduced up to a factor of almost 3 $\times$.

Zhang et al. [146] propose a signal processing-based secure big image data processing method through fog computing environment. We have considered this study in this category, since the main focus of the authors are about making the processing procedure secure. In the proposed method, a color image, represented by the discrete wavelet transformation is sampled and compressed using an advanced signal processing technology called compressive sensing. The sine logistic modulation map is employed to construct a measurement matrix used to perform the compressive sensing encoding. Furthermore, for security purposes authenticated measurements are also generated as encoding the color image. The generated measurements are normalized (for security purposes) and are sent to the fog nodes for image post-processing boosted with specific security mechanisms, i.e., extracting the value order, decomposing the measurements, and masking the energy value (based on a permutation-diffusion architecture to hide the energy information). Finally, the relevant data are transmitted to the data center for the purpose of storage, reconstruction, and integrity authentication. The authors have implemented the proposed processing in fog nodes in FPGA for the purpose of acceleration. The in-network computing is image post-processing boosted with security mechanisms which is implemented in DE2-70 FPGA. The proposed method is co-design since cloud data center is used for storage, reconstruction, and integrity authentication. Experimental results shows the privacy assurance of the proposed method under some attacks. Furthermore, using the proposed method the reconstruction time has been reduced up to a factor of 2 $\times$.

C. 4G/5G/6G

We first give a review of the studies that leverage in-network computing in Radio Access Network (RAN) of 4G/5G/6G. Then, we explain studies have been done with the focus on

mobile core network. Finally, we review the studies that have leveraged in-network computing in other areas of 4G/5G/6G.

1) Radio Access Network: In [147], an edge gateway facility has been deployed in RAN of LTE. The study in [148] offloads parts of next generation NodeB functionalities to programmable switches. Finally, the study in [149] improves the handover operation in RAN by leveraging in-network computing.

Aghdai et al. [147] propose an edge gateway in radio-access network of LTE that enables service operators deploy network functions at a close proximity to mobile users. They consider two functionalities as the major functionalities of the edge gateway: (i) content delivery to the users at the mobile edge; (ii) steering the received traffic to one of the MEC services while applying a load balancing strategy in traffic distribution. The in-network computation, i.e., aforementioned edge gateway functionality is implemented in Netronomr NFP4000 P4 target at the edge of IP transport. For a simple topology at which UE is connected via intermediate gateway to a node hosting SPGW, HSS, and MME components, the end-to-end delay of LTE protocol stack and the proposed gateway is in average 50 μs.

Vörös et al. [148] focus on implementation of 5G RAN Next generation NodeB (gNodeB) using programmable switches for their high throughput. Considering that some of gNodeB functionalities, e.g., ciphering/deciphering can not be implemented in programmable switches, a hybrid approach using programmable switch and external services is followed. Indeed, the main packet processing is implemented in a programmable switch while the complex functionalities are carried out by external services. The in-network computation, i.e., a fraction of functionalities of gNodeB is implemented in P4 hardware switch. The proposed method is a co-design approach since complex functionalities of gNodeB is served by services using general purpose processors. The evaluation on a P4 hardware switch demonstrates that the proposed hybrid approach could be an alternative to existing gNodeB solutions.

Palagummi and Sivalingamf [149] consider a next generation RAN, where the Base Band Unit functions are split across a Central Unit (CU) and multiple Distributed Units (DUs). The authors focus on a handover problem at which giving service to the mobile User Equipment (UE) needs handover between DUs. The authors propose a resource allocation at which resources are allocated ahead of the UE on its path. The functionality of the CU/DU is decomposed between three components including compute servers, P4 switches, and a controller. P4-based switches operates between the CU and the DUs. Among the functionalities of CU/DU, the in-network computation consists of tracking the mobility behaviour of UE and performing the resource allocation in DU, in advance, which are implemented in P4BM software switches. As a fraction of CU/DU functionalities are offloaded to the network, the proposed method is a co-design approach. Implementation shows that the proposed method have around 18% and 25% reduction in handover time.

2) Mobile Packet Core: The study in [150] describes a redesign for LTE EPC mobile packet core with offloading some control plane procedures to the programmable switch. The study in [151] proposes a gateway implemented in a programmable switch for 5G mobile packet core. Finally, [152] leverages in-network computing for implementing the user plane of Serving Gateway, while [153] proposes an in-network implementation for the User Plane Function.

The control plane of LTE EPC mobile packet core include procedures *attach*, *detach*, *S1 release*, *service request* and *handover*. Major of the signaling traffic is related to the procedures *S1 release*, and the *service request*, which manages the forwarding status of the user when it becomes idle or active. As these procedures operate on user-specific context, Shah et al. [150] propose offloading of these procedures into the packet processing pipeline of programmable data plane switches, thereby, improving throughput and latency in the control plane. Three challenges have also been considered: First, diverging of the state of control plane stored in switches from the master copy in the centralized control plane. To deal with this challenge, the state of the offloaded control plane is synchronized with its master copy. Second, to store the user context in the data plane, while respecting the memory limitation of the switches, user context is partitioned across multiple switches. Third, to deal with switch failures and avoid loss of the user context stored in switches, user context is replicated across switches, accordingly a mechanism to tackle switch failures is proposed. The in-network computation, i.e., *S1 release* and *service request* procedures in the control plane of LTE EPC is implemented by BMv2 and Netronome Agilio CX smartNICs. The proposed method is a co-design approach because the rest of LTE EPC control plane functionalities are implemented in non-network elements. The hardware prototype shows throughput and latency improvements by up to 102 $\times$ and 98% respectively when the switch hardware stores the state of 65K concurrent users.

Singh et al. [151] focus on 5G mobile packet core architecture. According to this architecture, the uplink and downlink IP traffic are routed to radio network eNodeB stations through the signaling gateway (SGW). Indeed, multiple eNodeBs and the handover between them is managed by SGWs. The connection between the mobile packet core and external IP networks, as well as functionalities like packet filtering, charging policies, and quality of service management are handled by Packet Data Network Gateway (PGW). On the other hand, the Mobility Management Entity (MME) performs security procedure, e.g., user authentication, session handling, and tracking of the user across the network.

The authors define Evolved Packet Gateway (EPG) that is a merge of the functions of both SGW and PGW. The authors implement vEPG user plane functions on a top of programmable switch, while keeping vEPG control plane on a x86 server. The pipeline to implement vEPG user plane include tables of L2 tables, firewall tables for uplink and downlink, GTP encapsulation and decapsulation table, and IPv4 routing tables. The in-network computation, i.e., user plane functionalities of SGW and PGW in mobile packet core architecture, is implemented in Barefoot Tofino hardware. The proposed method runs at line rate with latency less than 2 μs.

Shen et al. [152] describe a simplified architecture for packet processing at 5G. In this architecture, the Serving Gateway

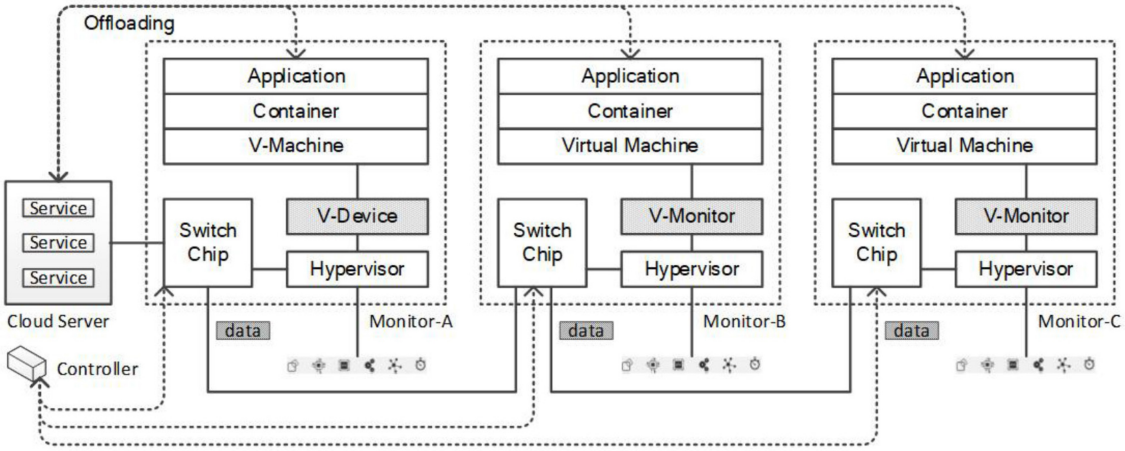

Fig. 10. In-Network computing architecture based on a virtualization platform [155].

(SGW) comprised of control plane (SGW-C) and user plane (SGW-U), at which several SGW-Us are controlled by one SGW-C. Backhaul provides connectivity among the based stations, core network, and edge network and the traffic transmission in the Backhaul is based on GPRS Tunneling Protocol (GTP).

GTP packets from the Backhaul are converted to Ethernet packets though SGW-U, before being sent to the servers at the edge. From the other hand, conversion of the Ethernet packets at the edge servers to the GTP packets before traversing toward the Backhaul is required. In this regard, a dedicated software/hardware is responsible for handling GTP headers. Since the software-based approach have low transmission throughput, the authors utilize programmable network elements to implement a SGW-U system for 5G mobile edge network. The in-network computation, i.e., SGW-U functionalities is implemented using Realtek RLT 9310 switch and FPGA platform. The experimental results show the throughput of 10 Gbps and the packet-processing latency of 5 μs have been achieved.

Paolucci et al. [153] provides a 5G X-haul testbed that has been enhanced with P4 switches implementing the User Plane Function (UPF) module. The in-network computation, i.e., UPF functionalities composed of GTP protocol encapsulation/decapsulation function, as well as N3-N6-N9 traffic steering in 5G MEC architecture. Furthermore, the monitoring of GTP flows performance metrics, e.g., the experienced latency, has also been implemented in the network through BMv2 and using the P4 code. Experienced latency for a flow of traffic is below 200 μs.

3) Other INC Applications: There are several studies that leverage in-network computing in other 4G/5G/6G applications. Processing of 5G slice flows and 6G tasks has been considered in respectively [154] and [155]. Ricart-Sanchez et al. [154] demonstrate an in-network solution which allows the processing of the flows in 5G network slices. They define a network slice by 6-tuple: 5G user source and destination IPs; 5G user source and destination ports; differentiated services code point; and GTP Tunnel ID which is related to a radio tunnel established between user equipment and 5G core network. The aforementioned 6-tuple are employed in a TCAM table in match/action stage, to define a network slice. The actions are defined to perform over slices. The network element pipeline has been extended to implement a queuing discipline to apply a priority-based discipline in processing the flows of the slices. According to the queuing discipline a low-priority queue will not be processed until the higher-priority queues have been processed. Indeed, several different queues are defined which allows various types of QoS for slices be defined. The in-network computation, i.e., the Processing of 5G slice flows is implemented in P4-NetFPGA. The evaluation results on a deployed 5G edge-to-core infrastructure in a private data centre, shows end-to-end latency under 0.5 ms for the highest priority queue and under 3 ms for the lowest priority queue in an end-to-end communication.

Hu et al. [155] propose an architecture, illustrated in Fig. 10 for applying in-network computing in 6G. The proposed architecture offloads network functions into the host systems equipped with switch chips at which packet processing function is implemented in a high-performance switching chip. In this regard, flexibility in application deployment as well as the capability of application migration will be provided. Indeed, a virtualization environment can be set up at the hosts, with the capability of application running in containers, which can be migrated among hosts according to a scheduling policy. The authors model the decision for offloading tasks operating over a data flow to the in-network computing devices as a multi-objective optimization problem that minimizes overall data transmission overhead, energy consumption, as well as idle rate of resources. An algorithm is proposed to solve the optimization. The in-network computing is task execution which is implemented through INC-Server contained a PX30Cortex- A35 CPU. For the a data aggregation applications the load on cloud server reduces up to 60% due to in-network processing and the energy consumption reduces up to 50%.

Wu et al. [156] focus on IoT applications that include several orders of magnitude higher connections in comparison with enhanced Mobile Broadband applications. From the other hand, the data traffic provided by each connected device is

much lower than the volume of traffic in legacy 4G communications. Considering the scenario of data transmission to multiple destinations, multiple small-data will be encoded into a chunk at a P4 switch before the data frames be transmitted. Then, through an eMBMS bearer, the chunk is transmitted from switch to the destination devices in the LTE-M cell. The decoding will be done at the destination after packet received at the IoT device. Using the proposed method the number of radio resource blocks used for data transmission has been reduced by 8 $\times$ in comparison with the benchmarks.

Gökarslan et al. [157] propose a programmable data plane for industrial 5G networks in P4, which reduces processing, provides a network monitoring mechanism, as well as enhancing the security of the network. The proposed data plane pipeline operates on the connection between RAN and User Plane Function (UPF). The routing decision for either sending GTP packets to the UPF or forwarding the packets to another gNodeB is offloaded to the P4 switches between gNodeBs. The authors also deploy monitoring and security functionalities at P4 pipeline to enhance the performance of industrial 5G. The in-network computation, i.e., decision about routing of traffic, monitoring, and security capabilities are implemented in BMv2. In comparison with the traditional 5G architecture, the proposed method reduces intra-cellular network latency up to 2 $\times$. Furthermore, security rules can be updated within 10 ms with a 95% confidence interval.

Ricart-Sanchez et al. [158] propose a hardware accelerated layer 4 firewall for 5G mobile networks. The proposed firewall operates between the edge and the core network in order to provide protection for 5G users, as well as the infrastructure. The firewall is implemented by adopting parser, match-action table, and deparser. Headers of MAC, IP, UDP/TCP and General Packet Radio Service Tunneling Protocol (GTP) are defined to be parsed. After the parsing, the packet will be processed by the match-action pipeline, where the extracted fields of the packet define a drop or forward action. A TCAM table is defined to include 5G user source/destination IPs and ports, transport protocol type, and GTP tunnel information as the match section. The DROP action will be applied for malicious packets while there is an allow-by-default policy. Finally, the non-dropped packets will be reconstructed and transmitted through the 5G infrastructure. The authors extended their study in [159], to support multi-tenancy in 5G. In-network computation, i.e., layer 4 firewall policies is implemented in a P4-NetFPGA NIC [160]. In the evaluations, the latency of packet processing in the network is 2493 times faster than a software-based solution and the throughput gain between edge and core nodes is up to 3.5 Gb/s.

D. Network Function Virtualization

Network Function Virtualization (NFV) is a paradigm, which decouples network functions from Application-Specific Integrated Circuits (ASICs) and specific hardware and implements them in virtualized infrastructures (e.g., virtual machines and containers). This effort results in handling of network functions, with lower cost and more flexibility for updating the functions. The deployment of virtual machines/containers consumes high resources, causes overheads by the operating system over the hypervisor, and furthermore the required performance and throughput for VNFs might not be provided. In-network computing has been leveraged to overcome these problems. The studies are categorized in two groups: Hardware-Accelerated Network Functions, and Framework/Deployment Solutions.

1) Hardware-Accelerated Network Functions: In order to diminish the performance issue when Network Functions (NFs) are running as software on top of common-off-the-shelf hardware, hardware acceleration techniques have been utilized for virtual network functions. NFs require tasks such as IP/MAC look-up for routing operations, encapsulation and decapsulation of packets for tunnel-based forwarding, encryption and decryption of packets for security. To perform such tasks frequent monitoring of NIC and processing the IP packet through the NF is required, which consumes high CPU cycles as well as I/O interactions. Hardware acceleration techniques which are categorized into custom and dedicated techniques can improve the efficiency of the process.

Dedicated hardware acceleration is designed in hardware for a specific function with limited or no capability of reprogramming and changing the behaviour of the hardware. Thus, dedicated hardware acceleration is not in the scope of in-network computing which demands the flexibility in general-purpose programming of network element. Custom acceleration is more cost efficient, programmable and configurable which lets to adopt new network functions and protocols depending on the application. The general idea behind these accelerators is to offload some processing to network element, e.g., FPGA, smart NIC, programmable switches, so that the processing can be applied on the packets before or after the processing performed by general CPU. In the context of network functions, checksum computations, encryption and decryption, splitting and rebuilding the packets [161], routing [162], load balancing [163] are examples of functionalities that can be offloaded to the network element to save CPU cycles and enhance the network function performance. We refer the interested readers to [45], [161] for more details.

2) Framework/Deployment Solutions: This category of studies propose framework and deployment solutions to exploit in-network computing in an NFV computing environment. While there is a category of studies that focus on deployment of VNFs in network elements, the other category of studies provide framework/deployment solutions in a NFV environment empowered with in-network computing.

a) Deployment of VNFs in network element: Kundel et al. [164], [165] use P4 and develop a Broadband Network Gateway data plane which is deployed in a Central Office Re-architected as a Datacenter (CORD) environment and provides the requirements of a telecommunication provider. The virtual network functions of Broadband Network Gateway system are discussed: traffic rate enforcement, customer tunneling, traffic access control, traffic separation, authentication, authorization and accounting, queing and hierarchical scheduling etc. The Broadband Network Gateway functions are performed in data plane on the upstream and downstream packets. The in-network

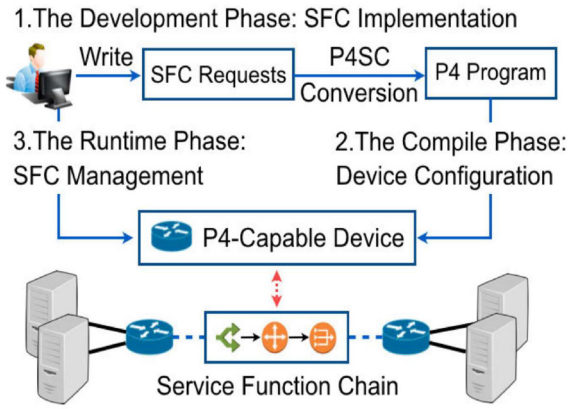

Fig. 11. A service function chaining framework on P4-enabled devices [170].

computation, i.e., Broadband Network Gateway functions is implemented in Barefoot Tofino, BMv2, Netronome SmartNIC, and P4-NetFPGA. For a scenario at which a subscriber is connected to core network via a subsequent of FPGA and P4 data plane (Tofino), and for 10000 packet VOIP transmission, the end-to-end latency is maximum 14.5 μs. Similarly, Osinski et al. [166] offload some functionalities of virtual Broadband Network Gateway to programmable ASIC, however there is no details or evaluation in their study.

Osinski et al. in [167], [168] propose a NFV framework in data centers that let the VNFs be instantiated in software switches or hardware devices like top of the rack switches, SmartNICs or FPGAs. A prototype of the framework is implemented based on OpenStack Neutron, P4 language and BMv2 software switch.

Mafioletti et al. [169] propose the deployment of VNFs on network elements, based on analyzing the functional components of the VNFs. The authors propose a framework to decompose VNFs into small embedded Network Functions (eNFs) to be deployed on network elements. The functional components of VNFs in a service function chain are examined to discover the repetitions in the processing of traffic. Accordingly, the common components in the VNFs are merged into a new component, according which the eNFs are defined to be offloaded to the network element. Finally, the corresponding P4 primitives including parsing and classification of headers and the actions required for packet dropping and counting, as well as the functionalities of eNFs are defined. A chaining mechanism based on a hash table and a bloom filter is also defined that allows to determine the network traffic direction in the chain. The in-network computation, i.e., small network functions is implemented on smart NIC. For a firewall VNF chain application with three VNFs, when all VNFs are run in the network, the latency can be reduced up to a factor of 76 $\times$ (43 μs) in comparison with the case that all VNFs are running at software (3300 μs). Throughput enhancement up to 8 $\times$ has been achieved.

The studies in [170], [171] design a framework for service function chaining (SFCs) on the P4-capable devices with the hardware capability as well as P4 programmability, to enhance performance, as well as flexibility in the implementation of Service Function Chains (SFCs). As shown in Fig. 11, the proposed framework offers several high-level primitives to the operators so that they could generate the required SFC requests. Furthermore, a converter generates the corresponding P4-program according to the given SFC requests. Converter applies an algorithm based on Longest Common Subsequence in order to merge multiple SFCs in a P4 program. The in-network computation, i.e., network functions is implemented on Tofino. The evaluation on real world SFC, illustrate that in comparison with software-based NFV solution, the throughput is enhanced up to a factor of $10^4\times$. Similarly the latency will be reduced up to a factor of $10^4\times$.

b) NFV empowered with INC (Framework/deployment solutions with hybrid substrate network): Deployment of VNFs over hybrid substrate network including network elements and general compute nodes has been considered in [172], [173]. Lopes et al. [172] propose a platform for the management and allocation of VNF components which are common across different VNFs, upon heterogeneous architectures comprised of FPGAs and general purpose processors. The authors give guidelines and discussion of properties of VNF components to help selecting the appropriate substrate for each VNF component. The in-network computation, i.e., VNF components is implemented on NetFPGA. The proposed method is a co-design approach because VNF components are deployed over a hybrid substrate environment including both network elements and general purpose processors. For deep packet inspection and firewall functions, in comparison with software solution, the proposed method has enhanced throughput by a factor of 2 $\times$, up to 800 Mbps.

An optimization framework for the deployment of VNFs on hybrid substrate network has been presented by Moro et al. [173]. The authors, decompose VNFs into several smaller functions, namely called μVNFs and distribute them over a hybrid infrastructure consisting of programmable switches, NICs, and edge/fog compute nodes. Then, the authors develop an optimization framework to select the decomposition with minimum cost of deployment, as well as identifying the node at which each μVNFs is deployed. The authors also deploy a tool through which multiple μVNFs are integrated within a single P4 program in order to be instantiated on a programmable switch. The robustness of the proposed algorithm in the condition of link failure is investigated. The in-network computation in the proposed method is μVNFs execution. The propose method is a co-design approach because the execution of μVNFs is distributed over a hybrid substrate environment including both network elements and edge/fog compute nodes. The proposed method has enhanced the deployment cost by 3 $\times$ in comparison with baselines.

E. Summary, Comparisons, Insights and Lessons Learned

In this section, we first briefly summarize the technology specific applications, and then discuss the insights and the lessons learned.

1) Summary: There are several studies that apply in-network computation in cloud computing. Works in [137], [138], [139], [140] leverage in-network computing

to provide load balancing in data centers. They offload load balancing to the network element. The studies in [27], [28], [141] provide resource allocation for data center applications in a cloud environment augmented with INC. At both [27], [28], network elements are utilized to perform application or application component computation. The study in [141] leverage in-network computing in a hybrid cloud provider and add-on providers.

There are also several studies in the scope of edge computing. Some studies apply in-network computing with the aim of edge intelligence provisioning. While [15] offload only critical subtasks of an intelligent recognition or controlling scenario to the network, the studies in [69], [81] exploit in-network computing for optimizations on federated learning. There are also some studies that provide resource allocation or task offloading methods in an edge computing empowered with in-network computing [58], [142], [143], [144]. Furthermore, the studies in [145] and [146] provide security mechanisms for respectively access control provisioning for massive machine communication and fog-based image processing.

Several studies have been done in the scope of 4G/5G/6G. The studies in [147], [148], [149] leverage in-network computing in Radio Access Network (RAN) to achieve higher throughout and lower latency. In these studies, computations like traffic steering, load balancing, some functionalities of gNodeB, and radio resource allocation have been offloaded to the programmable switches. Several studies, i.e., [150], [151], [152], [153] have leveraged in-network computing to facilitate mobile packet core. They have leveraged network elements to perform some control plane procedures, or offloadable mobile packet core network functions like Serving Gateway and User Plane Function. The studies in [154], [155], [156], [157], [158] have leveraged in-network computing for faster processing in other areas of 4G/5G/6G including 5G network slicing, 6G applications, LTE serving IoT application, and monitoring/securing 5G networks.

Network function virtualization suffer high consumption of resources due to the deployment of virtual machines/containers. Furthermore, there are overheads caused by the operating system over the hypervisor, that might end to not satisfaction of the required performance and throughput. In-network computing has been leveraged in several studies to overcome these problems. Hardware-accelerated network functions have been considered in surveys [45], [161]. There are studies that propose framework/deployment solutions to exploit in-network computing in NFV environment. Some studies focus on deployment of VNFs in network elements [164], [165], [167], [168], [169], [170], [171]. Generally, the proposed frameworks/deployment, suggest the execution of the whole VNF or a fraction of VNF functionality achieved by decomposing the VNF, be implemented in network elements, e.g., SmartNICs, FPGAs, programmable switches. The studies in [172], [173] provide framework/deployment solutions in a NFV environment empowered with in-network computing, i.e., hybrid of network elements and general-purpose computing nodes.

2) Comparisons, Insights and Lessons Learned: Tables XII, XIII, XIV, and XV compares the technology specific studies in the scopes of cloud, edge, 4G/5G/6G and NFV. The studies are compared from the aspect of the contribution, methodology (in-network computation, co-design, optimization/objective function), and evaluation (network element, platform in simulation, main results). There are some insights from which some lessons can be learned.

- *Fault Tolerance:* An overall insight is that few studies, e.g., [140], [150], have considered fault tolerance in technology-specific applications. Fault tolerance is crucial for in-network computing, since the network elements, particularly switches may fail. Furthermore, congestion might cause the network elements become unavailable. Considering 5G RAN as an example, when the execution of the offloaded RAN functionality to the switch fails due to switch failure, some potential malfunctions which is not acceptable for services with high availability requirement might happen: e.g., the connection between UE and the core network might be disconnected, the malfunctioning of handover. A lesson we learn is that more research effort is required to provide fault tolerance techniques for in-network computing.
- *Computing Environment Empowered with INC:* Another insight is that most of the studies focus on implementing required functions on network elements. However, infrastructure consists of heterogeneous resources including a hybrid of network elements and compute nodes with each have its own capability and property. While network elements provide considerable higher processing speed, they have less flexibility in comparison with compute nodes with powerful processing and memory capabilities. In this regard, an optimization is required to customize the trade-off and reach to an optimal decision. Few studies have followed the optimization over hybrid substrate network: [28] focuses on maximizing the admission of requests in a cloud empowered with INC, [173] minimizes VNF cost deployment in an NFV empowered with INC. We learn a lesson that more research effort is required to decide about optimal distribution of computation in an hybrid infrastructure, particularly in 5G/6G environment that lacks such optimization view in the existent studies.
- *Optimization:* The surveyed studies mostly consider exploitation of in-network computing through implementing the required functionality, e.g., 5G RAN, mobile packet core functionalities in new generations of mobile communications, virtual network functions, edge intelligence. However, providing the solutions to decide about optimal strategy for optimization of the system or application related performance criteria, in the targeted computing environment is required to efficiently exploit in-network computing besides the existent technologies. Few studies have considered this optimization approach, mostly in the scope of edge computing [58], [142], [145] with focuses on response time and resource utilization optimization. A learned lesson is that more research effort is required to provide solutions with the approach

TABLE XII
COMPARISON OF WORKS IN CLOUD COMPUTING. DS (DATA STRUCTURE), OBJ. FUNC. (OBJECTIVE FUNCTION), PLAT. (PLATFORM), Y (YES), N (NO), H (HADRWARE), S (SOFTWARE)

Scope	Ref.	Main Contribution	Methodology				Evaluation		
			In-network Computation	Co-Design	DS	Optimization/Obj. Func.	Network Element	Plat.	Main Results
Load Balancing	[137]	Proposing to use switching ASICs to build fast load balancers in data centers	Load balancing	N	Bloom filter; Hash Table	N	P4 switch ASIC	H	Throughput up to ten million connections at line rate
	[138]	Proposing a weighted equal-cost multi-path load-balancing scheme in data center networks.	Detecting flowlets; Estimating utilization of paths; Packet routing	N	Hash Table	N	BMv2	S	Up to 2% reduction in flow completion time
	[139]	Proposing a transport protocol targeting latency-sensitive RPC calls within a datacenter with the approach of load balancing	Load balancing policies i.e., Random, Round-Robin, JSQ and Join-Bounded-Shortest-Queue	N	-	N	Software router Tofino	H/S	Latency reduction by 5.7× Throughput incereamnet by more than 4.8×
	[140]	Proposing a combination of in-network computing and software implementation to provide a flexible, fast, and with high capacity load balancing method for data centers	Traffic splitting; Packet encapsulation	Y	Hash Table	N	Simulation based switch	S	By more than 10× latency reduction
Resource Allocation	[27]	Proposing a method that offloads application execution from data center server to network element in an online manner	Application execution	Y	Hash Table	N	NETFPGA Tofino	H	By 1000× power consumption reduction
	[28]	Proposing an resource allocation based on heuristic that allocates resources to maximum number of requests in a computational environment containing a hybrid of server and network resources	Application component computing	Y	-	Y (Allocate resources to maximum number of requests)	-	-	Up to 20% bandwidth saving Up to 50% placement latency reduction
	[141]	Leveraging in-network computing to provide resource allocation in an environment containing a hybrid of cloud provider and add-on providers	Distribute packets between add-on VMs and cloud services	N	-	N	-	-	-

of optimizing various system/application related criteria, for particularly technologies like NFV, cloud computing, and 5G/6G. This is particularly important for optimizing power consumption due to offloading computation from general purpose-computing nodes with high power consumption to network elements with lower power consumption, as the only study that considers this matter is [155]. Energy efficiency is particularly an important focus of design in 6G and providing power-efficient optimizations for resource allocation in 6G augmented with in-network computing, can be regarded as a future research trend.

In the rest, we provide more comparison from the aspect of computing node, performance metrics and methodology. Generally, the studies have applied in-network processing in these technologies to gain outperforming in QoS crieteria, e.g., throughput, and latency enhancement as well as improving resource-utilization criteria, e.g., power consumption and bandwidth consumption reduction, in comparison with existent server/dedicated hardware-based solutions. The main achieved outperforming in these criteria have been reported in Tables XII, XIII, XIV, and XV. For the purpose of better comparison, Table XVI compares the in-network implemented functionalities versus serve/dedicated hardware-based schemes, which are baselines for comparison in many studies, e.g., [157], [169], [170], [171], [172]. We provide more details as below:

- *Computing Node:* Server/Dedicated hardware-based solutions exploit general purpose computers or dedicated hardware (e.g., VNF middle wares, gNodeB) as computing nodes. In contrast, in in-network implemented schemes network elements play an important role as computing nodes, although exploiting a hybrid of network elements and other non-network element resources is also possible.
- Cost: In contrast with server/dedicated hardware-schemes, in-network implemented schemes, will reduce cost by managing the processing in already existed the data plane elements and omitting the necessity of high cost of computational resources, e.g., servers for load balancing [137], [140], or dedicated hardware, e.g., gNodeB in 5G [148] or specialized gateway, e.g., SGW and PGW in 5G [151].
- *Latency/Throughput:* In-network schemes can end to solutions with lower latency/higher throughput in comparison with server/dedicated-hardware based solutions due to high processing capabilities of network elements as well as proposing the computation closer to end-devices. Latency reduction by 10 × in comparison with a software-based cloud load balancer [140], up to 15 ×

TABLE XIII
COMPARISON OF WORKS IN THE SCOPE OF EDGE COMPUTING. OBJ. FUNC. (OBJECTIVE FUNCTION), PLAT. (PLATFORM), Y (YES), N (NO), H (HADRWARE), S (SOFTWARE)

Scope	Ref.	Main Contribution	Methodology: In-network Computation	Co-Design	Optimization/Obj. Func.	Network Element	Evaluation: Plat.	Evaluation: Main Results
Edge Intelligence	[15]	Proposing an architecture for mobile edge computing for industrial IoT which is based on offloading critical subtasks involved in provisioning of edge intelligence service to the network	Lightweight critical subtasks e.g., matrix multiplication in a robotic motion controlling application	Y	N	P4 switch	S	Accuracy 100%
	[81]	Proposing a federated learning method implemented by edge gateways and cloud aggregator in a multi-domain edge/cloud environment	Neural network	Y	N	BMv2	S	Accuracy 95%
	[69]	Proposing a federated learning method at edge based on an adopted concept of service-centric networking which lets the data be trained at workers close to data source with the capability of in-network computing	Caching and model training	N	N	Open vSwitch	S	-
Resource Allocation/Task Offloading	[58]	Proposing an architecture for fog services over a Wireless Mesh Network, as well as an in-network resource selection algorithm for services	Microservice execution	N	Y (selection of fog node with minimum response time	Mesh network element	S	Latency less than 6 ms
	[142]	Proposing a linear optimization for task offloading decision in an edge computing domain with SDN-enabled nodes	Task execution	N	Y (network usage minimization)	SDN-enabled node	S	Bandwidth saving by a factor of 10x
	[143]	Proposing an optimization framework for deciding about implementing hardware-acceleration in computational nodes as well as allocation of edge tasks to nodes	Task execution	Y	Y(Has defined only constraints)	FPGA	S	2× Latency reduction 44× Throughput increment 20× Energy reduction
	[144]	Proposing a FPGA-based accelerated method deployed at the network edge for the purpose of offloading compute-intensive tasks to accelerate mobile applications of hand-written digit number recognition, object recognition, and face detection.	neural network inference and computer vision algorithm operations	Y	N	ARMA-FPGA board (Xilinx Zc706)	H	Up to 15× latency reduction Up to 16% of energy reduction
Security	[145]	Proposing an access control method for massive machine communication with access control implementation at access equipment and securing the access control procedure by block chain.	calculating the statistics of accessing services and running access control policies	Y	Y (Optimal selection of policies)	BMv2	S	Up to 8 ms latency reduction Up to a factor of 3× bandwidth saving
	[146]	Proposing a secure big image data processing method through fog computing environment with in-network processing at fog-layer	image post-processing boosted with specific security mechanisms	Y	N	DE2-70 FPGA	H	Up to 2× latency reduction

latency reduction in mobile applications due to FPGA acceleration at edge [144], latency less than 6 ms for end-to-end message delivery through mesh relay/routing nodes playing the role of fog nodes [58], as well as latency of 50 μs for in-network LTE protocol stack and gateways processing [147], and up to 25% reduction in handover time between Distributed Units in next generation RAN [149], have been reported in the literature. As examples of throughput enhancement, the evaluations on real world service function chains in [170], [171] illustrate that software-based NFV solution will have roughly 0.01 Mbps throughput in software-based approach, while it will end to roughly 10^2 Mbps throughput utilizing the in-network implementation of network functions. As another example, up to 102 × throughput increcrement has been reported through offloading LTE EPC control plane operations to the programmable switches [150].

- *Power Consumption:* The processing capability of network elements per watt usage of energy is considerably higher than servers in cloud, edge, and NFV environment. For example, billions of operations is performed in programmable switches per watt usage. However, most of

TABLE XIV
COMPARISON OF WORKS IN THE SCOPE 4G/5G/6G. TECH. (TECHNOLOGY), OBJ. FUNC. (OBJECTIVE FUNCTION), PLAT. (PLATFORM), Y (YES), N (NO), H (HADRWARE), S (SOFTWARE)

Scope	Tech.	Ref.	Main Contribution	Methodology			Evaluation	
				In-network Computation	Co-Design	Optimization/Obj. Func.	Network Element	Plat.	Main Results
Radio Access Network	LTE	[147]	Proposing an edge gateway in radio-access network of LTE to deploy network functions at a close proximity to mobile users	Load balancing; Content delivery	N	N	Netronomr NFP4000 P4 target	H	Latency 50 μs
	5G	[148]	Proposing a hybrid approach using programmable switch and external services to implemente 5G RAN next generation NodeB functionalities	A fraction of functionalities of gNodeB	Y	N	P4 hardware switch	H	Throughput enhancement
	5G	[149]	Proposing a handover solution for mobile UE in RAN 5G environment with a focus on implementing the functionalities of the CU/DU through servers and P4 switches	Tracking the mobility of UEs; Performing the resource allocation in DU	Y	N	P4BM software switches	S	18% to 25% Handover time reduction
Mobile Packet Core	LTE	[150]	Proposing to offload LTE EPC control plane procedures that operate based on user-context to the programmable switches in the data plane	$S1$ $release$ and $service$ $request$ procedures in the control plane of LTE EPC	Y	N	BMv2; Netronome Agilio CX smartNICs	H	Up to 102× throughput increment 98% Latency decrements
	5G	[151]	Proposing a gateway through merging the functionalities of PGW and SGW in mobile packet core and accelerating gateway functionalities with in-network computing	User plane functionalities of SGW and PGW in mobile packet core architecture	N	N	Barefoot Tofino	H	Latency less than 2 μs
	5G	[152]	Utilizing programmable network elements to implement a SGW-U system for 5G mobile edge network	SGW-U functionalities	N	N	Realtek RLT 9310; FPGA	H	Throughput 10Gbps Latency of 5 μs
	5G	[153]	providing a 5G X-haul testbed that has been enhanced with P4 switches implementing the User Plane Function (UPF) module	UPF functionality; Monitoring of GTP flows	N	N	BMv2	S	Latency below 200 μs
Others	5G	[154]	Proposing an in-network solution for processing of the flows in 5G network slices	Processing of 5G slice flows	N	N	P4-NetFPGA	H	Up to 3 ms latency
	6G	[155]	Proposing a host-based architecture for in-network computing and developing an optimization model for offloading 6g tasks to in-network elements	Task execution	N	Y (minimize data transmission overhead, energy consumption, as well as idle rate of resources)	PX30Cortex-A35 CPU as INC-server	H	Up to 60% bandwidth saving Up to 50% energy saving
	LTE	[156]	Proposing an aggregation method for aggregating multiple small data frames using P4 Switches for IoT application in LTE cellular environment	Data encoding	N	N	P4 switch	S	Up to 8x radio resource block usage reduction
	5G	[157]	Proposing a novel data path for industrial 5G networks which is located between RAN and user plane and performs routing and security functionalities	Decision about routing traffic; Monitoring; Security capabilities	N	N	BMv2	S	Up to 2× latency reduction.
	5G	[158], [159]	Proposing a hardware accelerated layer 4 firewall for 5G mobile networks	Layer 4 firewall policies	N	N	NetFPGA NIC	H	2493× Latency reduction Up to 3.5 Gb/s throughput

the surveyed studies have not assessed in-network computing effect in power consumption and more research is required for this assessment. The study in [27], shows that through offloading cloud applications with high volume of message exchanges to the network or offloading a high power-consumed application from a server to the network, the power consumption is improved by 100 × in comparison with a software system on commodity CPU in cloud computing environment. Furthermore, energy saving up to 50% has been reported in [155] due to in-network processing of 6G tasks.

- *Disadvantages of In-network implemented Schemes:* In comparison with server/hardware-dedicated based solutions, there are two disadvantages: (i) Due to hardware limitations, in-network implemented solutions have less capability in implementing functions with complexity or high volume of data requirement. The functions can be application-level functions in NFV, cloud/edge, or new

TABLE XV
COMPARISON OF SOLUTION/DEPLOYMENT WORKS IN THE SCOPE OF NFV

Scope	Ref.	Main Contribution	Methodology In-network Computation	Co-Design	Optimization/Obj. Func.	Evaluation Network Element	Plat.	Main Results
Deployment of VNFs in Network Element	[164], [165]	Proposing a P4-based design and implementation of a Broadband Network Gateway data plane which runs in a Central Office Re-architected as a Datacenter	Broadband Network Gateway functions	N	N	Barefoot Tofino; BMv2; Netronome SmartNIC; P4-NetFPGA	H	Up to 14.5 μs latency
	[167], [168]	Proposing a framework to deploy VNFs inside network elements	VNF	N	N	-	-	-
	[169]	Proposing a framework to decompose virtual network functions into small network functions and deploy them on network elements	Small network functions	N	N	Smart NIC	H	Up to 76× latency decrement Up to 8× throughput increment
	[171] [170]	Proposing a framework for implementing Service Function Chains on the P4-capable devices	Network functions	N	N	Tofino	H	Up to $10^4\times$ throughput increment Up to $10^4\times$ latency decrement
Solutions With Hybrid Substrate Network	[172]	Proposing a platform for accelerating the execution of VNFs exploiting a hybrid hardware acceleration and software solutions	VNF components	Y	N	NetFPGA	H	Up to 2× throughput increment
	[173]	Proposing an optimization framework based on decomposing functionality of VNFs for the deployment of VNFs on hybrid substrate network	μVNF	Y	Y (VNF cost deployment minimization)	-	-	Up to 3× deployment cost reduction

TABLE XVI
COMPARISON OF FULLY-IN-NETWORK IMPLEMENTED FUNCTIONALITIES OF TECHNOLOGIES, WITH SERVER/DEDICATED HARDWARE-BASED SCHEMES

Scheme	Computing Node	Advantages	Disadvantages
In-Network Implemented Schemes	• Network Element • Hybrid with general purpose computers or dedicated hardware (e.g., VNF middle wares, gNodeB)	• Lower cost – Utilizing already existed the data plane elements – Omitting the necessity of high cost computational resources e.g., servers for load balancing, gNodeB in 5G, specialized gateway, SGW and PGW in LTE • Lower latency/higher throughput due to high processing capabilities of network elements as well as proposing the computation closer to end-devices • Lower power consumption than servers in cloud, edge, and NFV environment.	• Less capability in implementing complex functions • Less capability in implementing functions requiring high volume of data • More complex resource allocation strategy due to hybrid computational environment and high number of variables involved in optimization
Server/Dedicated Hardware-Based Schemes	• General purpose computers • Dedicated hardware e.g., gNodeB, VNF middleware	• Higher capability in implementing complex radio, core, and application-level functions	• Higher Cost • Higher Latency • Lower Throughput • Higher Power Consumption

generations of communications, as well as radio-access or core functions in 4G/5G/6G. Co-design schemes that utilize general-purpose computing units besides network element extend the capability of in-network solutions. The study in [15] is an example in robotic motion control application that offloads control matrix learning to MEC server, while performing the multiplication of matrix with current state vector in the programmable switches. As another example is the study in the application of machine communication [145], which implements access control policy at edge of the network in programmable switch and keeps the complex attack detection in the server. Considering the volume of data, partitioning of virtual-IP to direct-IP mapping elements for the purpose of load balancing in data centers among programmable switches is a strategy that has been used in [140] to cope with

TABLE XVII
COMPARISON OF SCHEMES FOR RESOURCE ALLOCATION IN HYBRID COMPUTATION ENVIRONMENT

Technique	**Operation**	Advantages	Disadvantages
Online	Dynamically allocate network elements or general purpose computing resources for computations at run time	• Capability to adopt the allocation at run time • Providing opportunity for switching between network elements and general purpose computating resources when performing computation • Higher capability for fault tolerance	Overhead at run-time decision
Offline	Allocate network elements or general purpose computing resources for computations at compile time	• Capability to find optimal allocation strategy • No overhead at run time	• Not to be able to adopt the allocation at run time • Not to be fault tolerant

memory limitation. Similarly in [150], user context to deal with LTE EPC mobile packet core procedures are divided among switches.(ii) Considering a hybrid computation environment including non-network resources (i.e., general computation resources in cloud, edge, NFV, 4G/5G/6G) as well as network elements, in comparison with server/dedicated hardware solutions, resource allocation decision will become more complex. The studies in [28] with the objective of allocating resources to maximum number of cloud applications, and [173] with the objective of VNF deployment minimization have suggested optimization techniques to cope with heterogeneity of resources. However, high dimensions of optimization variables can be involved in the problem, e.g., resources utilization of network elements, communication cost/time between network elements and general computational resources, processing capability of network elements etc. Due to high number of optimization variables, conventional optimization methods might not be efficient and investigation of more advanced optimization methods like machine learning can be a future potential research.

At the end of this section, we compare existent resource allocation schemes in aforementioned technologies empowered with INC. Among the surveyed studies, [27], [28], [58], [172], [173] perform resource allocation in an hybrid computation environment. They either perform the allocation in an online manner [27], [58] or offline manner [172], [173]. Table XVII compares the online and offline schemes. In online schemes resources are allocated dynamically at run time. One advantage is capability to adopt the allocation at run time as well as providing an opportunity for switching between network resources and conventional computing resources. As an example, the study in [27] provides an opportunity to dynamically switch the run of application between network element and cloud server, depending on parameters like power consumption of application and message exchange variations. Furthermore, online schemes propose higher capability for fault tolerance due to capability of changing the already allocated failed resource with another one. This is critical as network elements like switches are prone to failure or congestion. On the other hand, the disadvantage is the overhead of decision for resource allocation at run-time. In offline schemes, network elements and general-purpose computing resources are allocated to computations at compile time, having no overhead at run time, as well as offering the capability of finding optimal placement of computations. The disadvantage is not to have flexibility at run time and not to be fault tolerant in the case that a resource fails.

VIII. RESEARCH DIRECTIONS

In this section, we discuss the most important research directions, mainly derived from lessons learned from our literature review, that will be invaluable as in-network computing matures. We will have a separate section of Generic Research Directions which includes research directions that can be applied for various applications, i.e., in-network analytics, in-network caching, in-network coordination, in-network security, as well as technology-specific applications.

A. In-Network Analytics

In comparison with a host based analytics where the data is transmitted to a centralized host in order to be aggregated, in-network analytics can reduce the volume of traffic flow in the network as well as reducing analytics time. Several studies have shown the potential of in-network analytics in the areas of data aggregation, machine learning, and other types of analytics in the network, e.g., heavy flow detection, controlling, and query processing.

1) Machine Learning: Few studies have implemented machine learning in programmable network elements, and those studies are far from machine learning methods that are implemented on general purpose computers. Viable starting points for neural network implementation can be the studies in [79], [81]. However, [79] uses quantization to simplify the neural operations, and [81] uses binary weights and sign function as the activation function. As those simplifications can end to a lower accuracy, a research direction is to examine the feasibility of implementing neural networks in the network elements or a hybrid of network elements and general purpose computers, without any simplification in order to achieve lower inference latency without compromising accuracy. Considering

deep neural network, as a trend of research in machine learning, storing all neural network parameters in the network element might be impossible due to memory limitations. A research direction is to utilize the techniques suggested by studies to overcome the memory limitation of programmable data plane devices, and adapting the machine learning implementation accordingly. The study in [174] has proposed utilizing external memory with Remote Direct Memory Access (RDMA) facility to expand the memory in switches. The authors in [175] propose an architecture which let the flow tables of the top-of-rack switches be installed either in local memory or externally on multiple servers in the rack. External flow tables can be accessed with RDMA, which can be supported with either the RDMA-based network adapters or normal network adapters. Therefore, an interesting research avenue is to adapt the deep neural network implementation under memory extension mechanisms. Considering non-neural based learning methods, in-network implementation of decision tree has been done in [59], [117]. Implementing other non-neural based learning methods on network elements could be another research direction. The study in [78] can be a starting point for Naive Bayse and SVM. However, this study assumes that the calculations of required mathematical functions are being previously set up as look up tables inside the device and does not give any detail on such provisioning of calculations. In order to achieve a complete implementation of non-neural based learning methods, investigating the feasibility of implementing the required mathematical functions in the programmable network elements or a hybrid of network elements and general purpose computers, is required.

2) Co-Design Analytics: Due to processing and memory limitations in network elements, not every kind of analytics might be implementable in network elements. Furthermore, the required functionalities for a specific analytic might be difficult to be mapped if not impossible, to the network forwarding architectures like match-action tables. Co-design approaches that perform analytics utilizing a conjunction of network elements with non-network elements (e.g., servers, controllers) seems to be promising for in-network analysis. The functions or operations in the analytics that are infeasible to be implemented in network elements or have implementation complexity can be performed at common compute nodes (e.g., servers, controllers). Few studies exploit co-design approaches for analytics. An example is the study in [86] that proposes a co-design approach for query processing at which simpler operations are performed in data plane, while complex operations are performed by a stream processor. The co-design approaches can be an invaluable research direction particularly for machine learning analytics with complex discriminator functions or kernel based calculations, at which the implementation on network elements might be complex or infeasible.

B. In-Network Caching

The studies in the literature, e.g., [92], [93], [94], [95], [96], exploit in-network caching fabric to facilitate content/item access. The surveyed studies provide protocols for accessing and modifying the content/item. However, to have an efficient in-network caching fabric more criteria can be considered. For example, end-devices that initiate the requests for content/item might be mobile. In this regard the issue of re-configuring the data stored in network elements within in-network caching fabric might rise to provide the contents at the proximity of the end-devices. As in-network computing is in infancy state, our surveyed studies lack mobility coverage and reconfiguration solutions for in-network caching fabric. For a key-value based retrieval system, this problem becomes complex when a content might be composite, i.e., consists of several keys distributed among the network elements. Thus, a potential research direction would be provisioning optimization solutions and controlling mechanisms to decide about reconfiguration of data stored in in-network caches according to mobility patterns of end-devices.

C. In-Network Security

The in-network security has received much more attention from the research community than in-network analytics, caching, and coordination. However, as discussed in the lessons learned, there are still some research gaps that need to be filled. We discuss possible research directions, mainly derived from the lessons learned from our surveyed studies.

1) Machine Learning Based Attack Detection: Most of the attack detection methods in the literature are threshold based at which attack is detected through comparison of traffic statistics (e.g., SYN/ACK packets statistic, TCP/UDP packets and connections statistics, statistics about flows and subnets) with some thresholds. However, machine learning techniques can be implemented on network elements and can tackle the attack detection with a higher accuracy. A starting point is the study in [117] that uses random forest method to detect attacks in the network. Other machine learning methods, particularly neural networks whose implementation has shown to be feasible in network elements, can be investigated for detecting a variety of network attacks.

2) Co-Design Attack Detection/Mitigation: The surveyed studies in the scope of in-network security have shown promising results when programmable network elements are utilized to perform security related functionalities, e.g., estimating traffic statistics, calculating some traffic features, detecting the attack, and attack mitigation. However, each network element can establish a model for traffic distribution according to the local traffic it serves. The local traffic distribution can not effectively determine the parameters to detect the attack occurrence. Through applying a co-design approach, network elements can be utilized in conjunction with (SDN) controller in detecting and mitigating attacks. Controller can provide a global view to the established models by the network elements, accordingly it can decide about the attack detection parameters (e.g., thresholds) more accurately. Thus, a co-design approach with collaboration of controller and network elements to detect and mitigate network attacks can be an interesting research direction. The studies in [104], [109] have proposed co-design methods with the aim of including a global model for attack

mitigation. The study in [109] focuses on filtering spoofed traffic, while [104] focuses on DDoS attack mitigation. Co-design approaches to elaborate global models with collaboration of network elements and global entities like controller to mitigate attacks in the network can be investigated for a wide-range of other attacks in the network.

D. Technology Specific Applications

A quarter of studies have leveraged in-network computing for a specific technology including cloud/edge computing, 4G/5G, and network function virtualization. However, there are still some research gaps that need to be filled before in-network computing matures. In this section, we introduce these research directions.

1) Orchestration in a Hybrid Environment: In these environments, the network infrastructure offers a variety of heterogeneous resources including a hybrid of network elements (e.g., programmable switches and routers, FPGA, smart NICs) and other nodes (e.g., physical compute nodes, virtual machines, virtual network functions, storage nodes). The network elements can provide considerable high processing speeds at line rate and can operate on the packets on the path, omitting the necessity of long packet traversal. However, network elements have less flexibility in comparison with compute nodes with powerful processing and memory capabilities. Furthermore, depending on the required functionality, it might be complex or infeasible to implement every network functionality in the network elements. An invaluable research direction is therefore orchestrating various resources to handle the required application which demands resource allocation and traffic steering solutions to deploy the required applications. However, as discussed in the lessons learned, most of the studies focus on implementing required functions on network elements, e.g., [58], [147], [150], [152], [153], [165]. The study in [143] decides about implementing hardware-acceleration in computational edge nodes however, it does not optimize any specific objective function. The study in [28] provides resource allocation in a hybrid of network elements and cloud computing nodes with the objective of maximum requests admission. The study in [173] considers VNF decomposition in a hybrid infrastructure composed of network elements and NFV computing nodes with the objective of deployment cost minimization. Investigating orchestration approaches while considering other objectives, e.g., throughput maximization, bandwidth minimization, power consumption minimization and quality of service metrics (e.g., latency, reliability) deserves further studies and efforts. Furthermore, research is required to provide orchestration mechanisms suitable for 5G/6G systems empowered with in-network computing as the current state of the art lacks orchestration mechanisms.

2) Fault Tolerant In-Network Computation: Fault tolerance is crucial for in-network computing, since the network elements, particularly switches, may fail or become unavailable due to congestion. Failure of a switch hosting mobile packet core or 5G RAN can end to failure of the service. In an application like fire detection, where the fire detection and sensing alarm are offloaded to the switch as in [15], there could be catastrophic consequences as a result of switch failure. Thus, it is crucial to provide fault tolerance mechanisms for in-network computing applications. Among our surveyed studies in technological domains, few studies, e.g., [140], [150], have considered fault tolerance. More research effort is required to provide fault tolerance techniques for in-network computing applications.

3) Migration of Computation: Considering that the technologies of cloud, edge, 5G/6G and NFV serve services to end-devices, the mobility of end devices (e.g., vehicular in vehicular network, cell phones) demands solutions that let the computation be performed in the proximity of end-devices to meet the ultra-low latency requirement of applications. In this regard migration of computation/task among network elements due to end-device mobility can be beneficial to meet the required latency. As in-network computing is in infancy state, among our surveyed studies, migration of computation has not been investigated yet. Within the context of in-network computing the migration of computation in FPGA, NIC, or programming switch demands re-programming of the network element in destination. To avoid, such overhead of reprogramming, an initial attempt is the study in [155] that suggests treating hosts with virtualization environment and equipped with high performance switch chips (for the purpose of packet processing), as a kind of in-network computing nodes. However, the architecture in [155] is an abstract architecture and the paper does not provide detail discussion or any evaluation of the architecture. A potential research direction will be on providing in-network computing architectures and optimization solutions that facilitate migration of computations among in-network computing nodes.

E. Generic Research Directions

In this section, we discuss research directions that can be introduced independent of in-network computing application or a specific technology at which in-network computing is applied.

1) Collaborative In-Network Computing: Unlike common computers and servers, a challenge with network elements is the scarcity of resources, i.e., computational power and storage limitations. In this regard, designing solutions based on collaboration among switches can be regarded as a valuable research direction to cope with scarcity of resources. In a collaborative scenario, distribution of required functionality or data among network elements, as well as communication among switches to perform the required function or decision is expected. Many of our surveyed studies make efforts to implement the functionality that fits in a programmable network element and does not intrinsically demand collaboration, e.g., [119], [136], [139], [151]. However, in-network computing has the potential to be applied in scenarios with large space of functionalities at which fitting the required functionality in a single device is not possible. An example of such collaborative scenario is the study in [176], at which switches play the role of a DNS server. As the domain name space can be large and cannot fit into a single switch,

an iterative resolution is implemented by referring from one switch to another switch. A valuable research direction is to leverage in-network computing in a collaborative manner to solve the problems that require large space of functionalities. 5G/6G mobile packet core functionality provisioning or the service function chaining in NFV are examples of problems that demand a collection of functionalities, not all of them can fit into a single network element, and thus have the potential of leveraging the approach of collaborative in-network computing to meet the required functionalities.

2) Appropriate Network Element Selection: Various network elements are different in provisioning of computational power, storage capacity, and throughput [13]. Therefore, various criteria can be used to select an appropriate network element for a particular application, e.g., the type and number of computations that are required to be performed on a packet, the size of fan-out data after it was processed by the network element, the being statefull or stateless of the computation, and the size of the required data to be stored in the network element. Most of our surveyed studies in the scope of in-network computing have implemented the in-network computation with programmable switches and few studies consider implementation on either FPGAs or smart NICs, without discussing the reason for network element selection. Thus, a research direction can be analyzing the application requirement and devices capabilities to provide directions on an appropriate selection of the network element for a particular application. Starting points can be the studies in [13], [177]. While [177] explores the design spaces of seven state-of-the-art software switches and compares their performance under specific application of routing NFV traffic, [13] have a more general view. Indeed, [13] gives some guidelines to implement several applications based on a few application features, i.e., number of operations per packet, required storage per packet processing, and output size after packet processing. However, it lacks any evaluation to assess the provided guidelines. Finally, [13] does not talk about applications like in-network security, in-network coordination or technology specific applications.

3) In-Network Computing and Constraints of Network Devices: An important challenge in in-network computing application development is the constraints of network elements in terms of computation and memory. Current network elements like programmable switches support a limited number of arithmetic operations performed on non-floating values. An approach to deal with computation limitation is implementing co-designed approaches at which the complex computations are performed by general purpose processors, e.g., [15], [86]. Through a co-design approach, the latency will increase but still a fraction of computations will be performed with high performance in the network. Another approach is applying simplification and approximation techniques, e.g., [79], [81]. This approach trades off accuracy for keeping performance in the network. These techniques can be investigated more to cope with computation limitations.

The amount of data stored in network elements is limited by the size of the on-chip memory (e.g., in ranges from tens to hundreds of megabytes for programmable switches). Recently, some enhancement regarding memory limitation have been proposed. For example, [174] has expanded the memory capacity of switches through utilizing external memory with Remote Direct Memory Access (RDMA) facility. Expanding flow tables of the top-of-rack switches by installing them in servers of the rack has been proposed in [175]. In-network computing applications particularly, in-network caching and in-network analytics which their performance highly depends on the accessible memory, can be investigated in the future to be applied in such enhanced scenarios.

4) Performance Aspect of In-Network Computing: The studies in the literature have extensively illustrated the outperformance of in-network solutions in comparison with server/controller/software-based solutions particularly in terms of latency and throughput enhancement. However, the performance of network element (e.g., programmable switch, NIC) might vary under various traffic load and with various packet size. For example, in [169], it has been discussed that for a firewall VNF chain application with three VNFs, when the packet size is less than 1280 byte and all VNFs are offloaded to the smart NIC, the latency will be reduced by a factor of 76 $\times$ (from 3300 μs in software-based solution to 43 μs in network-based solution). However, when the packet sizes increase, packet processing at NIC can cause a negative impact on the latency measurements. Indeed, with 1518-byte packets, the latency enhancement between no offloading and full offloading is roughly 100 μs. In this situation, the authors in [169] discuss that it is appropriate to offload partial of VNFs to the network. However, still we have a research gap, to assess the benefit of in-network computing under various conditions of the network traffic and packet sizes in various applications which can be a potential research direction.

IX. Conclusion

Technologies of SDN, programmable data plane, edge computing, as well as protocols like information/service-centric networking enable In-Network Computing (INC). This manuscript provides the first survey on the topic of INC. The proposed categorization of studies comprises in-network analytics, in-network caching, in-network security, in-network coordination, and technology specific applications in the scopes of cloud/edge computing, 5G/6G, and NFV. The related studies are compared considering proposed methodology/implementation-related criteria, e.g., the performed computation in the network, following a fully-in-network implementation or a co-design approach, the utilized network element/data structure, as well as the performance gain in terms of latency, throughput, bandwidth saving, and power consumption. Furthermore, application specific criteria are proposed for comparison of approaches.

Our study shows that in comparison with server/SDN-controller based schemes, INC-based schemes will gain benefits considering application specific criteria. In-network analytics can gain higher inference speed, with less bandwidth consumption for data transmission. Either shallow/deep in-network caching schemes offer storage at edge, thereby reducing content access delay, bandwidth consumption for content

transmission, as well as providing an enhanced balancing of serving requests. In-network security schemes offer lower mitigation latency and reduce bandwidth consumption required for traffic analysis. In comparison with server/dedicated hardware-based schemes, in-network implementation of functionalities of technologies (e.g., RAN/mobile packet core functionalities of new generations of mobile communications, virtual network functions, edge intelligence), will reduce required computational cost and power consumption in implementation, while offering a lower experience of latency and higher throughput.

However, our study shows that there exist possible compromised application specific criteria due to hardware limitation of network elements. For each category of applications, the compromised criteria and the techniques to cope with hardware limitations have been extensively discussed in the survey. Furthermore, co-design approaches have been investigated as mechanisms to enrich INC with general purpose computation, and both the benefit gains and compromised application specific criteria in comparison with fully-in-network implemented schemes, have been discussed for various categories of applications. Complementary assessing of the resource allocation schemes for technology-specific applications to deal with hybrid computational environment have been provided. Finally, we discuss the potential research directions in this newly emerging topic.

REFERENCES

[1] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *J. Internet Services Appl.*, vol. 1, pp. 7–18, Apr. 2010.

[2] V. Ziegler et al., "6G architecture to connect the worlds," *IEEE Access*, vol. 8, pp. 173508–173520, 2020.

[3] M. Agiwal, A. Roy, and N. Saxena, "Next generation 5G wireless networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 1617–1655, 3rd Quart., 2016.

[4] "Amazon Cloud Service." [Online]. Available: https://aws.amazon.com/ec2/instance-types/

[5] C. Mouradian et al., "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 416–464, 1st Quart., 2018.

[6] A. J. Ferrer, J. M. Marquès, and J. Jorba, "Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing," *ACM Comput. Surveys*, vol. 51, no. 6, pp. 1–36, 2019.

[7] M. Mehrabi et al., "Device-enhanced MEC: Multi-access edge computing (MEC) aided by end device computation and caching: A survey," *IEEE Access*, vol. 7, pp. 166079–166108, 2019.

[8] B. A. A. Nunes et al., "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1617–1634, 3rd Quart., 2014.

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

[10] R. Bifulco and G. Rétvári, "A survey on the programmable data plane: Abstractions, architectures, and open problems," in *Proc. Conf. High Perform. Switch. Routing*, 2018, pp. 1–7.

[11] S. Han, S. Jang, H. Choi, H. Lee, and S. Pack, "Virtualization in programmable data plane: A survey and open challenges," *IEEE J. Commun. Soc.*, vol. 1, pp. 527–534, 2020.

[12] A. Sapio et al., "In-network computation is a dumb idea whose time has come," in *Proc. Conf. Hot Topics Netw.*, 2017, pp. 150–156.

[13] D. R. Ports and J. Nelson, "When should the network be the computer?" in *Proc. Conf. Hot Topics Oper. Syst.*, 2019, pp. 209–215.

[14] "Intel Second-Generation Programmable Switch." [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch

[15] T. Mai, H. Yao, S. Guo, and Y. Liu, "In-network computing powered mobile edge: Toward high performance industrial IoT," *IEEE Netw.*, vol. 35, no. 1, pp. 289–295, Jan./Feb. 2021.

[16] "Sigarch." [Online]. Available: https://www.sigarch.org/in-network-computing-draft/

[17] H. Stubbe, "P4 compiler & interpreter: A survey," *J. Future Internet Innov. Internet Technol. Mobile Commun.*, vol. 47, pp. 1–6, May 2017.

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

[19] X. Zhang, L. Cui, K. Wei, F. P. Tso, Y. Ji, and W. Jia, "A survey on stateful data plane in software defined networks," *J. Comput. Netw.*, vol. 184, Jan. 2021, Art. no. 107597.

[20] T. Dargahi et al., "A survey on the security of stateful SDN data planes," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1701–1725, 3rd Quart., 2017.

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

[22] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE Access*, vol. 9, pp. 87094–87155, 2021.

[23] M. G. Venkata, G. Bloch, G. Shainer, and R. Graham, "Accelerating OpenSHMEM collectives using in-network computing approach," in *Proc. Symp. Comput. Architect. High Perform. Comput.*, 2019, pp. 212–219.

[24] E. Bulut and M. Yuksel, "Integrating in-network computing for secure and efficient cascaded delivery in DTNs," in *Proc. Workshop Netw. Meets Intell. Comput.*, 2019, pp. 19–24.

[25] I. Kettaneh et al., "Falcon: Low latency, network-accelerated scheduling," in *Proc. P4 Workshop Europe*, 2020, pp. 7–12.

[26] F. Yang et al., "Understanding the performance of in-network computing: A case study," in *Proc. IEEE Conf. Parallel Distrib. Process. Appl. Big Data Cloud Comput. Sustain. Comput. Commun. Soc. Comput. Netw.*, 2019, pp. 26–35.

[27] Y. Tokusashi et al., "The case for in-network computing on demand," in *Proc. EuroSys Conf.*, 2019, pp. 1–16.

[28] M. Blöcher, L. Wang, P. Eugster, and M. Schmidt, "Switches for HIRE: Resource scheduling for data center in-network computing," in *Proc. ACM Conf. Archit. Support Program. Lang. Oper. Syst.*, 2021, pp. 268–285.

[29] R. L. Graham et al., "Scalable hierarchical aggregation protocol (SHARP): A hardware architecture for efficient data reduction," in *Proc. Workshop Commun. Optim. HPC*, 2016, pp. 1–10.

[30] T. A. Benson, "In-network compute: Considered armed and dangerous," in *Proc. Workshop Hot Topics Oper. Syst.*, 2019, pp. 216–224.

[31] P. G. Kannan and M. C. Chan, "On programmable networking evolution," *CSI Trans. ICT*, vol. 8, pp. 69–76, May 2020.

[32] A. Doria et al., "Forwarding and control element separation (ForCES) protocol specification," IETF, RFC 5810, 2010.

[33] "Floodlight, An Open SDN Controller." [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview

[34] "Beacon." [Online]. Available: https://openflow.stanford.edu/display/beacon/home

[35] M. R. Nascimento et al., "Virtual routers as a service: The routeflow approach leveraging software-defined networks," in *Proc. Conf. Future Internet Technol.*, 2011, pp. 34–37.

[36] A. Doria et al., "Forwarding and control element separation (ForCES) protocol specification," IETF, RFC 5810, 2010.

[37] "Barefoot Tofino: World's Fastest P4-Programmable Ethernet Switch ASICs." [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html

[38] "XPliant Ethernet Switch Product Family." [Online]. Available: https://www.openswitch.net/cavium/

[39] "Intel FlexPipe." [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ethernet-switch-fm6000-series-brief.pdf

[40] B. Pfaff et al., "The design and implementation of Open vSwitch," in *Proc. USENIX Symp. Netw. Syst. Design Implement.*, 2015, pp. 117–130.

[41] M. Shahbaz et al., "Pisces: A programmable, protocol-independent software switch," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 525–538.

[42] A. Panda et al., "NetBricks: Taking the V out of NFV," in *Proc. USENIX Symp. Oper. Syst. Design Implement.*, 2016, pp. 203–216.

[43] "BMv2 Software Switch." [Online]. Available: http://bmv2.org/
[44] N. Gebara et al., "Challenging the stateless quo of programmable switches," in *Proc. ACM Workshop Hot Topics Netw.*, 2020, pp. 153–159.
[45] P. Shantharama, A. S. Thyagaturu, and M. Reisslein, "Hardware-accelerated platforms and infrastructures for network functions: A survey of enabling technologies and research studies," *IEEE Access*, vol. 8, pp. 132021–132085, 2020.
[46] O. Michel, R. Bifulco, G. Retvari, and S. Schmid, "The programmable data plane: Abstractions, architectures, algorithms, and applications," *ACM Comput. Surveys*, vol. 54, no. 4, pp. 1–36, 2021.
[47] 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.
[48] A. Forencich, A. C. Snoeren, G. Porter, and G. Papen, "Corundum: An open-source 100-Gbps NIC," in *Proc. Symp. Field Program. Custom Comput. Mach.*, 2020, pp. 38–46.
[49] G. P. Katsikas, T. Barbette, M. Chiesa, D. Kostić, and G. Q. Maguire, "What you need to know about (SMART) network interface cards," in *Proc. Conf. Passive Active Netw. Meas.*, 2021, pp. 319–336.
[50] P. M. Phothilimthana et al., "FLOEM: A programming system for NIC-accelerated network applications," in *Proc. USENIX Symp. Oper. Syst. Design Implement.*, 2018, pp. 663–679.
[51] S. Choi, M. Shahbaz, B. Prabhakar, and M. Rosenblum, "λ-NIC: Interactive serverless compute on programmable smartNICs," in *Proc. Conf. Distrib. Comput. Syst.*, 2020, pp. 67–77.
[52] "P4: Language Specification." [Online]. Available: https://opennetworking.org/wp-content/uploads/2020/10/P416-language-specification.html#fig-p4prg
[53] 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.
[54] A. Sivaraman et al., "Packet trans.: High-level programming for line-rate switches," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 15–28.
[55] C. J. Anderson et al., "NetKAT: Semantic foundations for networks," *ACM Sigplan Notices*, vol. 49, no. 1, pp. 113–126, 2014.
[56] P. Bosshart et al., "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.
[57] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct.–Dec. 2009.
[58] S. Ali, M. Pandey, and N. Tyagi, "Wireless-fog mesh: A framework for in-network computing of microservices in semipermanent smart environments," *J. Netw. Manag.*, vol. 30, no. 6, 2020, Art. no. e2125.
[59] X. Zhang, L. Cui, F. P. Tso, and W. Jia, "pHeavy: Predicting heavy flows in the programmable data plane," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4353–4364, Dec. 2021.
[60] Y. Afek, A. Bremler-Barr, and L. Shafir, "Network anti-spoofing with SDN data plane," in *Proc. Conf. Comput. Commun.*, 2017, pp. 1–9.
[61] T.-Y. Lin et al., "Mitigating SYN flooding attack and ARP spoofing in SDN data plane," in *Proc. Asia–Pac. Netw. Oper. Manag. Symp.*, 2020, pp. 114–119.
[62] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, and K.-K. R. Choo, "P4-to-blockchain: A secure blockchain-enabled packet parser for software defined networking," *J. Comput. Security*, vol. 88, Jan. 2020, Art. no. 101629.
[63] A. Morrison, L. Xue, A. Chen, and X. Luo, "Enforcing context-aware BYOD policies with in-network security," in *Proc. USENIX Workshop Hot Topics Cloud Comput.*, 2018, pp. 1–9.
[64] Q. Kang et al., "Programmable in-network security for context-aware BYOD policies," in *Proc. USENIX Security Symp.*, 2020, pp. 595–612.
[65] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A local search mechanism for peer-to-peer networks," in *Proc. Conf. Inf. Knowl. Manag.*, 2002, pp. 300–307.
[66] Q. Lv et al., "Search and replication in unstructured peer-to-peer networks," in *Proc. Conf. Supercomput.*, 2002, pp. 84–95.
[67] M. Gritter and D. R. Cheriton, "An architecture for content routing support in the Internet," in *Proc. Symp. Internet Technol. Syst.*, vol. 1, 2001, pp. 37–48.
[68] G. Xylomenos et al., "A survey of information-centric networking research," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 1024–1049, 2nd Quart., 2013.
[69] X. Li, R. Xie, F. R. Yu, T. Huang, and Y. Liu, "Advancing software-defined service-centric networking toward in-network intelligence," *IEEE Netw.*, vol. 35, no. 5, pp. 210–218, Sep./Oct. 2021.
[70] A. Nasrallah et al., "Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 88–145, 1st Quart., 2018.
[71] R. Ahmed and R. Boutaba, "A survey of distributed search techniques in large scale distributed systems," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 2, pp. 150–167, 2nd Quart., 2010.
[72] A. Biswas, S. Mohan, and R. Mahapatra, "Optimization of semantic routing table," in *Proc. Conf. Comput. Commun. Netw.*, 2008, pp. 1–6.
[73] R. F. Martins, F. L. Verdi, R. Villaça, and L. F. U. Garcia, "Using probabilistic data structures for monitoring of multi-tenant P4-based networks," in *Proc. IEEE Symp. Comput. Commun.*, 2018, pp. 204–207.
[74] F. Yang et al., "SwitchAgg: A further step towards in-network computing," in *Proc. IEEE Conf. Parallel Distrib. Process. Appl. Big Data Cloud Comput. Sustain. Comput. Commun. Soc. Comput. Netw.*, 2019, pp. 36–45.
[75] A. L. R. Madureira, F. R. C. Araújo, and L. N. Sampaio, "On supporting IoT data aggregation through programmable data planes," *J. Comput. Netw.*, vol. 177, Aug. 2020, Art. no. 107330.
[76] C. Li and H. Dai, "Efficient in-network computing with noisy wireless channels," *IEEE Trans. Mobile Comput.*, vol. 12, no. 11, pp. 2167–2177, Nov. 2013.
[77] C. Li et al., "Towards efficient designs for in-network computing with noisy wireless channels," in *Proc. IEEE INFOCOM Conf.*, 2010, pp. 1–8.
[78] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? Toward in-network classification," in *Proc. ACM Workshop Hot Topics Netw.*, 2019, pp. 25–33.
[79] D. Sanvito, G. Siracusano, and R. Bifulco, "Can the network be the AI accelerator?" in *Proc. Workshop In Netw. Comput.*, 2018, pp. 20–25.
[80] Y.-S. Lu and K. C.-J. Lin, "Enabling inference inside software switches," in *Proc. Asia–Pac. Netw. Oper. Manag. Symp.*, 2019, pp. 1–4.
[81] Q. Qin, K. Poularakis, K. K. Leung, and L. Tassiulas, "Line-speed and scalable intrusion detection at the network edge via federated learning," in *Proc. Netw. Conf.*, 2020, pp. 352–360.
[82] V. Sivaraman et al., "Heavy-hitter detection entirely in the data plane," in *Proc. Symp. SDN Res.*, 2017, pp. 164–176.
[83] R. Harrison, Q. Cai, A. Gupta, and J. Rexford, "Network-wide heavy hitter detection with commodity switches," in *Proc. Symp. SDN Res.*, 2018, pp. 1–7.
[84] J. Vestin, A. Kassler, and J. Åkerberg, "FastReact: In-network control and caching for industrial control networks using programmable data planes," in *Proc. Conf. Emerg. Technol. Factory Autom.*, vol. 1, 2018, pp. 219–226.
[85] F. E. R. Cesen et al., "Towards low latency industrial robot control in programmable data planes," in *Proc. IEEE Conf. Netw. Softw.*, 2020, pp. 165–169.
[86] A. Gupta et al., "Sonata: Query-driven streaming network telemetry," in *Proc. ACM Special Interest Group Data Commun.*, 2018, pp. 357–371.
[87] R. Teixeira, R. Harrison, A. Gupta, and J. Rexford, "PacketScope: Monitoring the packet lifecycle inside a switch," in *Proc. Symp. SDN Res.*, 2020, pp. 76–82.
[88] T. Jepsen et al., "Life in the fast lane: A line-rate linear road," in *Proc. Symp. SDN Res.*, 2018, pp. 1–7.
[89] T. Kohler et al., "P4CEP: Towards in-network complex event processing," in *Proc. Workshop In Netw. Comput.*, 2018, pp. 33–38.
[90] J. Hypolite et al., "DeepMatch: Practical deep packet inspection in the data plane using network processors," in *Proc. Conf. Emerg. Netw. Exp. Technol.*, 2020, pp. 336–350.
[91] Y. Tokusashi, H. Matsutani, and N. Zilberman, "LAKE: The power of in-network computing," in *Proc. Conf. Reconfig. Comput. FPGAs*, 2018, pp. 1–8.
[92] Q. Wang et al., "Concordia: Distributed shared memory with in-network cache coherence," in *Proc. USENIX Conf. File Storage Technol.*, 2021, pp. 277–292.
[93] X. Jin et al., "Netcache: Balancing key-value stores with fast in-network caching," in *Proc. Symp. Oper. Syst. Principle*, 2017, pp. 121–136.
[94] M. Liu et al., "Incbricks: Toward in-network computation with an in-network cache," in *Proc. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2017, pp. 795–809.
[95] Z. Liu et al., "Distcache: Provable load balancing for large-scale storage systems with distributed caching," in *Proc. USENIX Conf. File Storage Technol.*, 2019, pp. 143–157.
[96] X. Jin et al., "NetChain: Scale-free sub-RTT coordination," in *Proc. USENIX Symp. Netw. Syst. Design Implement.*, 2018, pp. 35–49.

[97] V. Jacobson et al., "Networking named content," in *Proc. Conf. Emerg. Netw. Exp. Technol.*, 2009, pp. 1–12.

[98] D. Scholz, S. Gallenmüller, H. Stubbe, and G. Carle, "SYN flood defense in programmable data planes," in *Proc. P4 Workshop Europe*, 2020, pp. 13–20.

[99] F. Musumeci et al., "Machine-learning-assisted DDoS attack detection with P4 language," in *Proc. Conf. Commun.*, 2020, pp. 1–6.

[100] J. Xing, W. Wu, and A. Chen, "Architecting programmable data plane defenses into the network with FastFlex," in *Proc. ACM Workshop Hot Topics Netw.*, 2019, pp. 161–169.

[101] Â. C. Lapolli, J. A. Marques, and L. P. Gaspary, "Offloading real-time DDoS attack detection to programmable data planes," in *Proc. Symp. Integr. Netw. Service Manag.*, 2019, pp. 19–27.

[102] L. A. Q. González et al., "BUNGEE: An adaptive Pushback mechanism for DDoS detection and mitigation in P4 data planes," in *Proc. Symp. Integr. Netw. Manag.*, 2021, pp. 393–401.

[103] X. Z. Khooi, L. Csikor, D. M. Divakaran, and M. S. Kang, "DIDA: Distributed in-network defense architecture against amplified reflection DDoS attacks," in *Proc. Conf. Netw. Softw.*, 2020, pp. 277–281.

[104] K. Friday, E. Kfoury, E. Bou-Harb, and J. Crichigno, "Towards a unified in-network DDoS detection and mitigation strategy," in *Proc. IEEE Conf. Netw. Softw.*, 2020, pp. 218–226.

[105] M. Zhang et al., "Poseidon: Mitigating volumetric DDoS attacks with programmable switches," in *Proc. Conf. NDSS*, 2020, pp. 1–9.

[106] Z. Liu et al., "JAQEN: A high-performance switch-native approach for detecting and mitigating volumetric DDoS attacks with programmable switches," in *Proc. USENIX Security Symp.*, 2021, pp. 37–48.

[107] X. Yang, J. Cao, and M. Xu, "SEC: Secure, efficient, and compatible source address validation with packet tags," in *Proc. Perform. Comput. Commun. Conf.*, 2020, pp. 1–8.

[108] H. Gondaliya, G. C. Sankaran, and K. M. Sivalingam, "Comparative evaluation of IP address anti-spoofing mechanisms using a P4/NetFPGA-based switch," in *Proc. P4 Workshop Europe*, 2020, pp. 1–6.

[109] G. Li et al., "NETHCF: Enabling line-rate and adaptive spoofed IP traffic filtering," in *Proc. IEEE Conf. Netw. Protocols*, 2019, pp. 1–12.

[110] J. Bai, J. Bi, M. Zhang, and G. Li, "Filtering spoofed IP traffic using switching ASICs," in *Proc. ACM SIGCOMM Conf. Posters Demos*, 2018, pp. 51–53.

[111] P. Kuang, Y. Liu, and L. He, "P4DAD: Securing duplicate address detection using P4," in *Proc. IEEE Conf. Commun.*, 2020, pp. 1–7.

[112] M. Dimolianis, A. Pavlidis, and V. Maglaris, "A multi-feature DDoS detection schema on P4 network hardware," in *Proc. Conf. Innov. Clouds Internet Netw. Workshops*, 2020, pp. 1–6.

[113] P. Vörös et al., "T4P4S: A target-independent compiler for protocol-independent packet processors," in *Proc. Conf. High Perform. Switch. Routing*, 2018, pp. 1–8.

[114] C. Monsanto et al., "Composing software defined networks," in *Proc. USENIX Symp. Netw. Syst. Design Implement.*, 2013, pp. 1–13.

[115] P. Vörös and A. Kiss, "Security middleware programming using P4," in *Proc. Conf. Human Aspects Inf. Security Privacy Trust*, 2016, pp. 277–287.

[116] R. Datta, S. Choi, A. Chowdhary, and Y. Park, "P4Guard: Designing P4 based firewall," in *Proc. IEEE Mil. Commun. Conf.*, 2018, pp. 1–6.

[117] J.-H. Lee and K. Singh, "SwitchTree: In-network computing and traffic analyses with random forests," *J. Neural Comput. Appl.*, vol. 2020, pp. 1–12, May 2020.

[118] T. Datta, N. Feamster, J. Rexford, and L. Wang, "SPINE: Surveillance protection in the network elements," in *Proc. USENIX Workshop Free Open Commun. Internet*, 2019, pp. 1–9.

[119] D. Chang, W. Sun, and Y. Yang, "A SDN proactive defense mechanism based on IP transformation," in *Proc. Conf. Safety Produce Informatization*, 2019, pp. 248–251.

[120] G. Liu et al., "P4NIS: Improving network immunity against eavesdropping with programmable data planes," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2020, pp. 91–96.

[121] J. Xing, A. Morrison, and A. Chen, "NetWarden: Mitigating network covert channels without performance loss," in *Proc. USENIX Workshop Hot Topics Cloud Comput.*, 2019, pp. 1–7.

[122] A. Laraba et al., "Defeating protocol abuse with P4: Application to explicit congestion notification," in *Proc. Netw. Conf.*, 2020, pp. 431–439.

[123] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems," in *Proc. Mil. Commun. Inf. Syst. Conf.*, 2015, pp. 1–6.

[124] A. El Kamel, H. Eltaief, and H. Youssef, "On-the-fly (D)DoS attack mitigation in SDN using deep neural network-based rate limiting," *J. Comput. Commun.*, vol. 182, pp. 153–169, Jan. 2022.

[125] H. T. Dang et al., "NetPaxos: Consensus at network speed," in *Proc. ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, 2015, pp. 1–7.

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

[127] H. T. Dang et al., "P4xos: Consensus as a network service," *IEEE/ACM Trans. Netw.*, vol. 28, no. 4, pp. 1726–1738, Aug. 2020.

[128] J. Li, E. Michael, N. K. Sharma, A. Szekeres, and D. R. Ports, "Just say NO to Paxos overhead: Replacing consensus with network ordering," in *Proc. USENIX Symp. Oper. Syst. Design Implement.*, 2016, pp. 467–483.

[129] Y. Zhang, B. Han, Z.-L. Zhang, and V. Gopalakrishnan, "Network-assisted RAFT consensus algorithm," in *Proc. SIGCOMM Posters Demos*, 2017, pp. 94–96.

[130] M. Kogias and E. Bugnion, "HovercRaft: Achieving scalability and fault-tolerance for microsecond-scale datacenter services," in *Proc. Conf. Comput. Syst.*, 2020, pp. 1–17.

[131] E. Sakic et al., "P4BFT: A demonstration of hardware-accelerated BFT in fault-tolerant network control plane," in *Proc. ACM SIGCOMM Posters Demos*, 2019, pp. 6–8.

[132] E. Sakic, N. Deric, E. Goshi, and W. Kellerer, "P4BFT: Hardware-accelerated Byzantine-resilient network control plane," in *Proc. IEEE Global Commun. Conf.*, 2019, pp. 1–7.

[133] S. Han, S. Jang, H. Lee, and S. Pack, "Switch-centric Byzantine fault tolerance mechanism in distributed software defined networks," *IEEE Commun. Lett.*, vol. 24, no. 10, pp. 2236–2239, Oct. 2020.

[134] Z. Yu et al., "Netlock: Fast, centralized lock management using programmable switches," in *Proc. ACM Special Interest Group Data Commun. Appl. Technol. Archit. Protocols Comput. Commun.*, 2020, pp. 126–138.

[135] J. Li, E. Michael, and D. R. Ports, "ERIS: Coordination-free consistent trans. using in-network concurrency control," in *Proc. Symp. Oper. Syst. Principle*, 2017, pp. 104–120.

[136] Z. István, D. Sidler, G. Alonso, and M. Vukolic, "Consensus in a box: Inexpensive coordination in hardware," in *Proc. USENIX Symp. Netw. Syst. Design Implement.*, 2016, pp. 425–438.

[137] R. Miao et al., "SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs," in *Proc. ACM Special Interest Group Data Commun.*, 2017, pp. 15–28.

[138] J.-L. Ye, C. Chen, and Y. H. Chu, "A weighted ECMP load balancing scheme for data centers using P4 switches," in *Proc. Conf. Cloud Netw.*, 2018, pp. 1–4.

[139] M. Kogias et al., "R2P2: Making RPCs first-class datacenter citizens," in *Proc. USENIX Annu. Tech. Conf.*, 2019, pp. 863–880.

[140] R. Gandhi et al., "DUET: Cloud scale load balancing with hardware and software," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 27–38, 2014.

[141] Z. Wu and H. V. Madhyastha, "Rethinking cloud service marketplaces," in *Proc. Workshop Hot Topics Netw.*, 2016, pp. 134–140.

[142] G. Lia et al., "Optimal placement of delay-constrained in-network computing tasks at the edge with minimum data exchange," in *Proc. IEEE 5G World Forum*, 2021, pp. 481–486.

[143] R. A. Cooke and S. A. Fahmy, "A model for distributed in-network and near-edge computing with heterogeneous hardware," *J. Future Gener. Comput. Syst.*, vol. 105, pp. 395–409, Apr. s2020.

[144] C. Xu et al., "The case for FPGA-based edge computing," *IEEE Trans. Mobile Comput.*, vol. 21, no. 7, pp. 2610–2619, Jul. 2022.

[145] F. Song et al., "Smart collaborative contract for endogenous access control in massive machine communications," *IEEE Internet Things J.*, early access, Dec. 10, 2021, doi: 10.1109/JIOT.2021.3134366.

[146] Y. Zhang et al., "Privacy-assured FogCS: Chaotic compressive sensing for secure industrial big image data processing in fog computing," *IEEE Trans. Ind. Informat.*, vol. 17, no. 5, pp. 3401–3411, May 2021.

[147] A. Aghdai, M. Huang, D. Dai, Y. Xu, and J. Chao, "Transparent edge gateway for mobile networks," in *Proc. Conf. Netw. Protocols*, 2018, pp. 412–417.

[148] P. Vörös, G. Pongrácz, and S. Laki, "Towards a hybrid next generation NodeB," in *Proc. P4 Workshop Europe*, 2020, pp. 56–58.

[149] P. Palagummi and K. M. Sivalingam, "SMARTHO: A network initiated handover in NG-RAN using P4-based switches," in *Proc. Conf. Netw. Service Manag.*, 2018, pp. 338–342.

[150] R. Shah, V. Kumar, M. Vutukuru, and P. Kulkarni, "TurboEPC: Leveraging dataplane programmability to accelerate the mobile packet core," in *Proc. Symp. SDN Res.*, 2020, pp. 83–95.

[151] S. K. Singh, C. E. Rothenberg, G. Patra, and G. Pongracz, "Offloading virtual evolved packet gateway user plane functions to a programmable ASIC," in *Proc. Workshop Emerg. In Netw. Comput. Paradigms*, 2019, pp. 9–14.
[152] C.-A. Shen et al., "A programmable and FPGA-accelerated GTP offloading engine for mobile edge computing in 5G networks," in *Proc. Conf. Comput. Commun. Workshops*, 2019, pp. 1021–1022.
[153] F. Paolucci et al., "User plane function offloading in P4 switches for enhanced 5G mobile edge computing," in *Proc. Conf. Design Rel. Commun. Netw.*, 2021, pp. 1–3.
[154] R. Ricart-Sanchez, P. Malagon, J. M. Alcaraz-Calero, and Q. Wang, "P4-NetFPGA-based network slicing solution for 5G MEC architectures," in *Proc. Symp. Archit. Netw. Commun. Syst.*, 2019, pp. 1–2.
[155] N. Hu, Z. Tian, X. Du, and M. Guizani, "An energy-efficient in-network computing paradigm for 6G," *IEEE Trans. Green Commun. Netw.*, vol. 5, no. 4, pp. 1722–1733, Dec. 2021.
[156] X. Wu, Z. Jin, W.-K. Jia, and X. Shi, "Aggregating multiple small-data frames using arithmetic encoding in P4 switches," in *Proc. Annu. Consum. Commun. Netw. Conf.*, 2021, pp. 1–6.
[157] K. Gökarslan, Y. S. Sandal, and T. Tugcu, "Towards a URLLC-aware programmable data path with P4 for industrial 5G networks," in *Proc. IEEE Conf. Commun. Workshops*, 2021, pp. 1–6.
[158] R. Ricart-Sanchez, P. Malagon, J. M. Alcaraz-Calero, and Q. Wang, "Hardware-accelerated firewall for 5G mobile networks," in *Proc. Conf. Netw. Protocols*, 2018, pp. 446–447.
[159] R. Ricart-Sanchez et al., "NetFPGA-based firewall solution for 5G multi-tenant architectures," in *Proc. Conf. Edge Comput.*, 2019, pp. 132–136.
[160] J. W. Lockwood et al., "NetFPGA—An open platform for Gigabit-rate network switching and routing," in *Proc. Conf. Microelectron. Syst. Educ.*, 2007, pp. 160–161.
[161] L. Linguaglossa et al., "Survey of performance acceleration techniques for network function virtualization," *Proc. IEEE*, vol. 107, no. 4, pp. 746–764, Apr. 2019.
[162] C. Fernández, S. Giménez, E. Grasa, and S. Bunch, "A P4-enabled RINA interior router for software-defined data centers," *J. Comput.*, vol. 9, no. 3, p. 70, 2020.
[163] H. Ballani et al., "Enabling end-host network functions," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 493–507, 2015.
[164] R. Kundel et al., "P4-BNG: Central office network functions on programmable packet pipelines," in *Proc. Conf. Netw. Service Manag.*, 2019, pp. 1–9.
[165] K. Ralf et al., "OpenBNG: Central office network functions on programmable data plane hardware," *J. Netw. Manag.*, vol. 31, no. 1, 2021, Art. no. e2134.
[166] T. Osiński et al., "Unleashing the performance of virtual BNG by offloading data plane to a programmable ASIC," in *Proc. P4 Workshop Europe*, 2020, pp. 54–55.
[167] T. Osiński, H. Tarasiuk, L. Rajewski, and E. Kowalczyk, "DPPx: A P4-based data plane programmability and exposure framework to enhance NFV services," in *Proc. Conf. Netw. Softw.*, 2019, pp. 296–300.
[168] T. Osiński, M. Kossakowski, H. Tarasiuk, and R. Picard, "Offloading data plane functions to the multi-tenant cloud infrastructure using P4," in *Proc. Symp. Archit. Netw. Commun. Syst.*, 2019, pp. 1–6.
[169] D. R. Mafioletti et al., "PIaFFE: A place-as-you-go in-network framework for flexible embedding of VNFs," in *Proc. Conf. Commun.*, 2020, pp. 1–6.
[170] D. Zhang et al., "P4SC: A high performance and flexible framework for service function chain," *IEEE Access*, vol. 7, pp. 160982–160997, 2019.
[171] X. Chen et al., "P4SC: Towards high-performance service function chain implementation on the P4-capable device," in *Proc. Symp. Integr. Netw. Service Manag.*, 2019, pp. 1–9.
[172] F. B. Lopes, G. L. Nazar, and A. E. Schaeffer-Filho, "VNFAccel: An FPGA-based platform for modular VNF components acceleration," in *Proc. Symp. Integr. Netw. Manag.*, 2021, pp. 250–258.
[173] D. Moro, G. Verticale, and A. Capone, "A framework for network function decomposition and deployment," in *Proc. Conf. Design Rel. Commun. Netw.*, 2020, pp. 1–6.
[174] D. Kim et al., "Generic external memory for switch data planes," in *Proc. ACM Workshop Hot Topics Netw.*, 2018, pp. 1–7.
[175] Y. Xue and Z. Zhu, "Hybrid flow table installation: Optimizing remote placements of flow tables on servers to enhance PDP switches for in-network computing," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 1, pp. 429–440, Mar. 2021.
[176] J. Woodruff, M. Ramanujam, and N. Zilberman, "P4DNS: In-network DNS," in *Proc. Symp. Archit. Netw. Commun. Syst.*, 2019, pp. 1–6.
[177] T. Zhang et al., "Performance benchmarking of state-of-the-art software switches for NFV," *J. Comput. Netw.*, vol. 188, Apr. 2021, Art. no. 107861.

Somayeh Kianpisheh received the Ph.D. degree in computer engineering from Tarbiat Modares University. From 2018 to 2020, for a period of two years, she was a Postdoctoral Researcher with Concordia University, Canada. She also has a Postdoctoral Research experience with Aalto University, Finland, in 2021. Since 2022, she has been a Postdoctoral Researcher with the University of Oulu, Finland. Her research interests include in-network computing, programmable data plane, SDN, edge computing, NFV, fog/cloud systems, and 5G and beyond.

Tarik Taleb received the B.E. degree (with Distinction) in information engineering and the M.Sc. and Ph.D. degrees in information sciences from Tohoku University in 2001, 2003, and 2005, respectively.

He is currently a Professor with the Centre for Wireless Communications–Networks and Systems Unit, Faculty of Information Technology and Electrical Engineering, The University of Oulu. He is the Founder and the Director of MOSA!C Lab. From October 2014 to December 2021, he was a Professor with the School of Electrical Engineering, Aalto University, Finland. Prior to that, he was working as a Senior Researcher and a 3GPP Standards Expert with NEC Europe Ltd., Heidelberg, Germany. Before joining NEC and until March 2009, he worked as an Assistant Professor with the Graduate School of Information Sciences, Tohoku University, Japan, in a Lab fully funded by KDDI, the second largest mobile operator in Japan. From October 2005 to March 2006, he worked as Research Fellow with Intelligent Cosmos Research Institute, Sendai, Japan. He has been also directly engaged in the development and standardization of the Evolved Packet System as a member of 3GPP's System Architecture working group 2. His research interests lie in the field of telco cloud, network softwarization and network slicing, AI-based software defined security, immersive communications, mobile multimedia streaming, and next generation mobile networking.

Prof. Taleb is the recipient of the 2017 IEEE ComSoc Communications Software Technical Achievement Award in December 2017 for his outstanding contributions to network softwarization and the 2021 IEEE ComSoc Wireless Communications Technical Committee Recognition Award in December 2021. He is also the co-recipient of the Young Researcher's Encouragement Award from the Japan chapter of the IEEE Vehicular Technology Society in October 2003, the Niwa Yasujirou Memorial Award in February 2005, the 2006 IEEE Computer Society Japan Chapter Young Author Award in December 2006, the 2007 Funai Foundation Science Promotion Award in April 2007, the 2008 TELECOM System Technology Award from the Telecommunications Advancement Foundation in March 2008, the 2009 IEEE ComSoc Asia-Pacific Best Young Researcher Award in June 2009, and the 2017 IEEE Communications Society Fred W. Ellersick Prize in May 2017. He served as the the Chair of the Wireless Communications Technical Committee, the largest in IEEE ComSoC until December 2016, and the General Chair of the 2019 Edition of the IEEE Wireless Communications and Networking Conference (WCNC'19) held in Marrakech, Morocco. He also served as the Vice Chair of the Satellite and Space Communications Technical Committee of IEEE ComSoc from 2006 to 2010. He was the Guest Editor-in-Chief of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS Series on Network Softwarization and Enablers. He was on the editorial board of the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, *IEEE Wireless Communications Magazine*, IEEE JOURNAL ON INTERNET OF THINGS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, and a number of Wiley journals. Some of his research work have been also awarded Best Paper Awards at prestigious IEEE-Flagged Conferences.