INVITED PAPER

Reconfigurable Network Systems and Software-Defined Networking

This paper reviews the current state of the art in reconfigurable network systems, covering hardware reconfiguration and its interplay with software-defined networking (SDN).

By Noa Zilberman, *Senior Member IEEE*, Philip M. Watts, *Member IEEE*, Charalampos Rotsos, *Member IEEE*, and Andrew W. Moore, *Member IEEE*

ABSTRACT | Modern high-speed networks have evolved from relatively static networks to highly adaptive networks facilitating dynamic reconfiguration. This evolution has influenced all levels of network design and management, introducing increased programmability and configuration flexibility. This influence has extended from the lowest level of physical hardware interfaces to the highest level of network management by software. A key representative of this evolution is the emergence of software-defined networking (SDN). In this paper, we review the current state of the art in reconfigurable network systems, covering hardware reconfiguration, SDN, and the interplay between them. We take a top-down approach, starting with a tutorial on software-defined networks. We then continue to discuss programming languages as the linking element between different levels of software and hardware in the network. We review electronic switching systems, highlighting programmability and reconfiguration aspects, and describe the trends in reconfigurable network elements. Finally, we describe the state of the art in the integration of photonic transceiver and switching elements with electronic technologies, and consider the implications for SDN and reconfigurable network systems.

KEYWORDS | Field-programmable gate array (FPGA); reconfigurable devices; software-defined networks; switching fabrics

I. INTRODUCTION

The Internet provides the infrastructure upon which our modern world is built. Computer networks underpin modern commerce and industry as well as enable the social networks that are at the heart of modern life. They are characterized by a continuous evolution, with tensions between the practical and the desirable. As a flourishing and fertile networking environment, the Internet has required innovative design and management practices to evolve. Into this environment, software-defined networks have come to describe a paradigm for exploring innovation in network design and operation. While software-defined networking (SDN) seems to have appeared suddenly, it is actually part of a long history of trying to make computer networks more programmable and to capitalize on the reconfigurability of the underlying systems.

It is our contention for this paper that SDN and its predecessors are distinctive from reconfigurable networks yet serve to drive the evolution of reconfigurable network systems. We maintain that the approach of the SDN paradigm will dominate the entire breadth of network system reconfigurability: from the configuration of devices at set-up to the reconfiguration and update of those devices over their lifetime. The SDN paradigm can offer well-defined

Manuscript received August 15, 2014; revised February 16, 2015 and April 27, 2015; accepted May 12, 2015. Date of publication June 11, 2015; date of current version June 18, 2015. This work was jointly supported by the U.K. Engineering and Physical Sciences Research Council (EPSRC) under Internet Project EP/H040536/1 and by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under Contract FA8750-11-C-0249. The work of P. M. Watts was supported by the Engineering and Physical Sciences Research Council (EPSRC) under the Research Fellowship Grant EP/I004157/2. The views, opinions, and findings contained herein are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of DARPA or the U.S. Department of Defense.

N. Zilberman and **A. W. Moore** are with the University of Cambridge, Cambridge CB3 OFD, U.K. (e-mail: noa.zilberman@cl.cam.ac.uk; andrew.moore@cl.cam.ac.uk).
P. M. Watts is with the University College London, London WC1E 6BT, U.K. (e-mail: philip.watts@ucl.ac.uk).
C. Rotsos is with the University of Lancaster, Lancaster LA1 4YW, U.K. (e-mail: c.rotsos@lancaster.ac.uk).

Digital Object Identifier: 10.1109/JPROC.2015.2435732

interfaces to devices. While such SDN interfaces may offer a subset of the devices' capabilities, they permit a flexible reconfiguration of network systems independently of the details of the device implementation. As network systems become more complex both in routine operation and in their configurations, SDN provides an evolvable pathway between devices.

To understand the symbiotic relationship between SDN and reconfigurable network systems, this paper takes a top–down approach. We begin with a short tutorial on software-defined networking. A discussion of the interfaces between different elements of a software-defined network follows, and Section III surveys programming languages used across these interfaces. We consider how SDN affected the evolution of programming languages over time, and extend the discussion to proprietary environments. Section IV provides a hardware perspective to reconfiguration in current electronic switching devices. This section describes header processing (as the main operation affected by SDN), and extends to additional networking functions as a place for future innovation in SDN. The technologies for reconfigurable network systems are introduced in Section V, and we predict the impact of these technologies upon such systems. Finally, Section VI considers the integration of photonic transceiver and switching elements with electronic technologies in future systems and discusses the implications for SDN and reconfigurable network systems. We appreciate that this paper targets a wide audience, and therefore suggest that SDN experts skip the remainder of this section along with Sections II and III.

A. Foundations of SDN

SDN is a network paradigm. As SDN is an assembly of ideas drawn from a range of innovation efforts, it has several slightly different definitions (e.g., [1]–[4]). Key to all these definitions are the strong isolation between different planes in the network (primarily between the control and data plane, as we explain next), central management, and a high level of programmability. The isolation between control plane and data plane is not unique to SDN; the accumulation of related functionality into layers is common practice across networking disciplines. The strength of isolation has varied across different types of networks.

In order to explain SDN, let us consider the common current network environment, the Internet. At its heart, the Internet consists of routers connected with each other and with hosts, servers, and clients. Routers form the nodes of a network interconnecting multiple hosts and other routers. Each router forwards packets along links, and the router decides where to forward packets in the data plane using information derived by the control plane. A simple control plane is a computer program that (among other duties) uses routing protocols to discover pathways upon which to forward packets. It had been commonplace for a single manufacturer to provide an integrated system that would implement a given set of routing protocols, compute appropriate forwarding rules, and install these in the hardware of the data plane. Such systems provide little opportunity to install new or experimental control planes (e.g., a new algorithm that routes packets according to a different criteria) but in return the limited flexibility was balanced by a system that offered most customers what they required. Each router vendor would satisfy the compliance need, meaning their products behaved correctly and conformed with appropriate Internet standards. However, such a vertically integrated system offered little opportunity for innovation. Additionally, the vertically integrated systems meant that a superior data plane from one vendor could not be simply connected to the superior control plane of a competitor vendor.

Such vertically integrated systems meant that network service providers (and researchers) were frustrated by the time and expertise needed to develop and deploy new network services. The widespread use of vertically integrated networking equipment has left limited opportunity for innovation. Aside from customers and researchers that wished to deploy and reinvent the control plane, researchers focused upon the network data plane realized that commercial systems rarely provided the right environment to evaluate their ideas. In contrast to the software used for implementing new routing algorithms in the control plane, innovation in the data plane could require measurements, and redesigned control of the data plane's high-speed networking hardware. Given that data-plane design is a delicate balance of considerations (speed, features, and pricing), unnecessary features were shunned by commodity network-equipment vendors. This lack of opportunity for innovation motivated the development of SDN on reconfigurable systems.

B. Reconfigurable Systems

The reconfiguration of hardware has been a core requirement underlying many decades of networking success. Reconfiguration in network systems covers a wide spectrum of use cases from the onetime configuration of devices when a system starts its life, through the runtime reconfiguration of algorithms implemented in networking devices to allow their operation at line rate, to adapting the operation of previously configured devices by maintenance programming.

Throughout much of the long history of computer networking, reconfigurable logic has provided core functionality in commodity electronics. Such an example is the early use of programmable array logic to permit the programming of unique device identifiers [e.g., media access control (MAC) address] after device manufacture. This permitted the cheap manufacture, assembly, and testing of devices—despite each one being uniquely configured.

Field-programmable gate array (FPGA) devices have a long association with high-speed networking equipment. For example, FPGA devices are commonly used to provide

the control logic required to interconnect application-specific integrated circuit (ASIC) devices while providing management functions along with power and thermal control. FPGA devices have been widely used on high-speed network interfaces to implement rules intended to process incoming and outgoing packets (e.g., for filtering packets) and continue to see use as offload processing, configured to provide data processing as either a coprocessor alongside more general purpose computer systems or a less-flexible dedicated network switch silicon. Since FPGA devices can be configured in live systems, they have seen widespread use implementing network protocols, and protocol-translation services. In each of these cases, the FPGA can be reconfigured as requirements change.

In this paper, we treat reconfiguration as a spectrum of activities ranging from initial configuration, through in-system changes in functional design (complete and partial reconfiguration), to include updating parameters within a reconfigurable device (such as table entries). We distinguish between reconfiguration and programmability. By reconfiguration we refer to selecting one option from a given set (including a range of values, e.g., set register value to `0xF`) to change the operation of an element. By programmability, we refer to providing a set of instructions (of varying type, number, and order, e.g., repeat a lookup operation until a match is found) to set the operation of an element.

C. Biased History: Reconfigurable Network Systems and SDN

We assert a close relationship between reconfigurable (network) systems and SDN. Recently, SDN owes much of its journey into popular consciousness on the back of OpenFlow [5], an interface between the control plane and the data plane. However, the core ideas of software-defined networking predate that work by several decades. Reconfigurability has played many critical roles, from the earliest implementations of network prototypes [6] through work on active networks [7] and flexible network systems [8]. In addition, a series of SDN interface approaches propose alternative protocols for control-plane programmability, and we present them extensively in Section III.

The OpenFlow interface was first and foremost an open-source standard. The code, documentation, and reference implementation (software and hardware) are openly available to any interested party. Such practice is not new, as much of the early Internet Engineering Task Force (IETF) work began as open development. What makes the OpenFlow particularly interesting was its reliance on open-source hardware based upon reconfigurable systems [9]. If SDN owes much of its relaunch to OpenFlow, then OpenFlow owes much of its popular adoption to the ready availability of implementations. The NetFPGA [10]–[12] platform, itself an open-source FPGA-based reconfigurable platform, provided the ideal base for a prototype OpenFlow hardware implementation [5].

It is clear that reconfigurable systems have made core contributions to networking in general and SDN in particular. Feamster *et al.* [2] provide a technical history of SDN, and additional surveys can be found in [3] and [13].

D. Scope and Related Work

We presume that the reader is familiar with the key elements of common Internet-style packet-switched networks, where each packet has information sufficient for conveyance toward the final destination. However, any network is more than a simple process that forwards packets along links in a network. Networks are subject to a combination of requirements such as the coordination of decisions about where to send packets, the need to optimally interconnect different types of physical networks (e.g., wireless mobile and wired), or the need to subdivide a network based on geography or administrative domain. Each function adds complexity to the organization and operation of the underlying network. Tackling the combinatorial effect of complexity is not specifically considered within the Internet. It is the control of this expansive complexity that SDN attempts to tackle.

We have purposely limited the scope of this paper to local area network (LAN) and wide area network (WAN) applications. While the intersection of reconfigurable systems and SDN is wide ranging, the resurgence of interest in SDN has remained firmly focused upon LAN and WAN (with emphasis on Internet protocols and wired Ethernet-based networks, unless otherwise noted). While this is not an SDN-imposed restriction, it is the most common use case for SDN-based networks.

Similarly, we limit the scope of this paper to wired networks. SDN is independent of the physical media, as the abstraction of the network elements make this implicit. However, we will not explicitly explore the implications for mobile/wireless networks here. Interested readers may find relevant an early SDN-enabled for mobile networks [14], optimizing placement of overlapping LTE cells [15], and an example of a full enterprise deployment incorporating authentication, authorization, and accounting [16]. Additionally, we mention only in passing the plethora of other software-controlled networks, such as advances in networks-on-chip (NoC) reconfiguration under software control. Such work has ranged from reconfigurable topologies [17] and configurable channels [18], to fault recovery [19] and circuit-switched NoCs [20]. Finally, while reconfigurable systems have made an impact upon the wireless domain through software-defined radio (SDR) [21], we will not discuss these technologies here.

While not specifically SDN, as a matter of scope it is important to make clear the relationship with network functions virtualization (NFV). NFV is an emerging network architecture concept that employs host virtualization technologies such as Xen [22] that allow entire classes of network node functions to be treated as building blocks. These blocks may be connected, or (in the language of

NFV) chained, together to create communication services. It is clear that SDN complements the NFV idea and provides a powerful enabling tool. However, NFV is a mechanism for organizing elements along the datapath in a network (e.g., firewalls, network intrusion detection systems, caches for various traffic types). Since an initial white paper [23], researchers have extracted some of the ideas of NFV into the form of a reconfigurable network system [24], and there is a nascent community engaged in abstracting such data-plane elements to enable SDN-like innovation in network function control [25].

II. SOFTWARE-DEFINED NETWORK

A. Introducing the Software-Defined Network

Utilizing SDN, the software-defined network may naively be considered as based upon simple concepts: SDN networks routinely utilize a common and well-defined interface between a control plane and a data plane. The control plane is responsible for global coordination (such as routing and fault recovery). The data plane is where routine packet-by-packet operations occur. Exceptional events in the data plane become events sent to the control plane. The control plane can modify data-plane behavior. The separation of the control plane and the data plane is not a new concept. It might be argued that what SDN popularized is the use of a clearly defined interface between the two. Next, we compare a traditional router with SDN.

Fig. 1 illustrates a router-based network alongside its SDN equivalent. In each case, there is a subdivision of work between the data plane and the control plane. The data plane implements several processing functions on each packet: 1) buffering (and/or storing) packets while the headers are processed; 2) examining the header and looking up header information in the forwarding table (the table storing the forwarding rules) to identify the actions the switch should perform; and 3) queuing packets for transmission. An example of a switch action is updating the time-to-live (TTL) in IP packets. (The TTL field is decremented each time a packet passes through a router. When the value reaches zero, the packet is discarded and an error is generated.) These functions are done for all routers, whether SDN or not.

In both cases, the control plane must handle all possible circumstances, including any exceptional packets. For example, in IPv4 and IPv6, packets that exceed their TTL require such exceptional handling to include discarding the packet and returning a control packet to the source reporting the error. The control plane also configures the data plane and manages the mechanisms by which the data

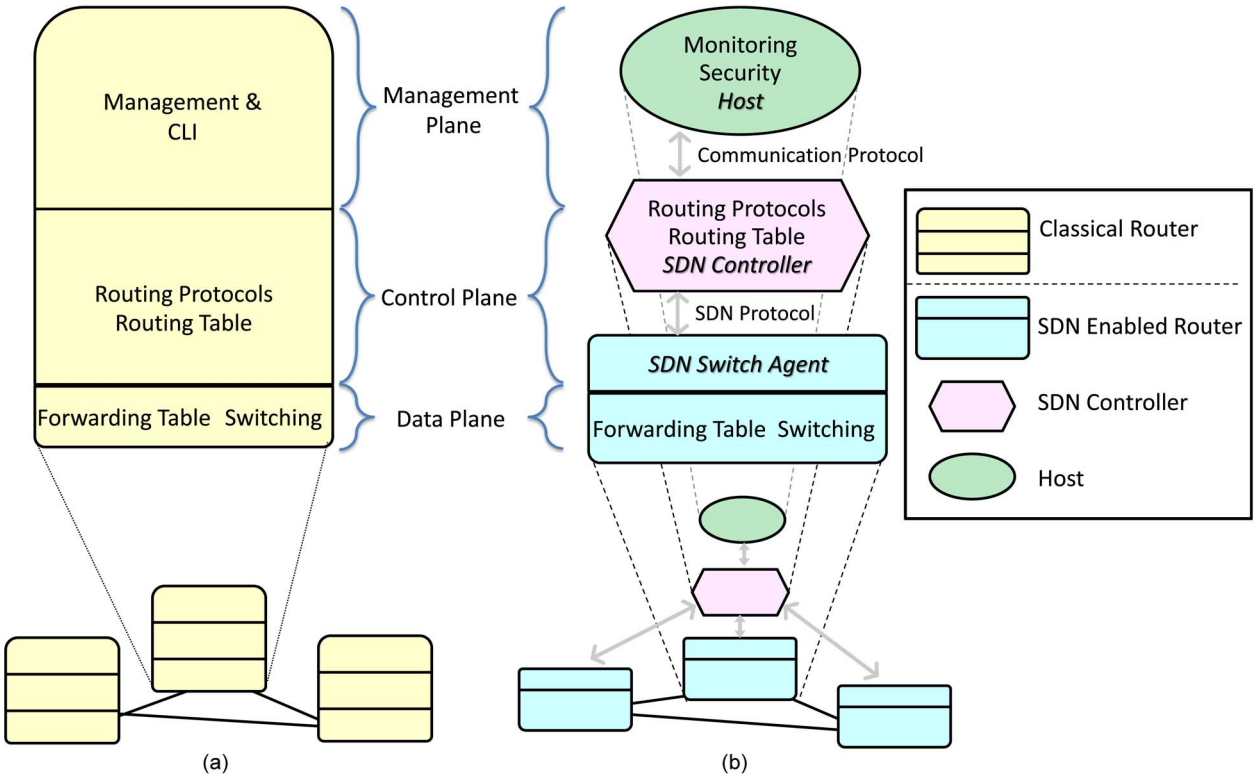

Fig. 1. *Functionality in a classical router-based network and the equivalent SDN network. (Hosts are not shown.) (a) Classical-router network. (b) SDN network.*

plane forwarding tables are computed and configured. Typically, this involves running one or more routing protocols, exchanging local routing information with other router peers, and deriving a local forwarding table.

Commonplace approaches to separate control-plane and data-plane functionality involve implementing control-plane functionality in a high-level programming language and operating on a general purpose processor. The control plane may implement sophisticated programs, but without optimization, per-packet processing performance may be low. In contrast, the data plane implements optimum network performance (e.g., high per-packet processing rate, minimal latency), but only for the most commonly encountered cases.

As show in Fig. 1, equivalent SDN systems support identical functionality. These consist of a data plane with an optimization for high-speed forwarding and a control plane to handle exceptions and create content for the forwarding table of a switch element. The differences arise due to differences in the abstractions defined between the data-plane system and the control-plane system among SDN implementations. Effectively, SDN treats network devices as fast but simplistic forwarding elements which can be used as building blocks for higher order functionality, such as routing and access control. Furthermore, by providing a common abstraction, new architectures can arise. For example, as illustrated in Fig. 1, the SDN network may share a single SDN controller among different switch elements.

B. Reducing Complexity: Motivation for SDN

Network architects, engineers, and operators are presented with the challenge to provide state-of-the-art network infrastructure and services, all while minimizing the associated purchase and operation costs. Researchers in networking extend this challenge by also seeking to explore novel and potentially disruptive ideas in a state-of-the-art network infrastructure. It is into this space that SDN has arisen.

A core design principle of the SDN paradigm is to define an open interface, exposed by network devices, which allows control of the connectivity and traffic flow of the device [26]. This interface definition allows seamless network reconfiguration of the network control logic. Effectively, SDN tries to define a common abstraction which encodes the reconfiguration capabilities of the underlying network devices.

The SDN thesis is in two parts: first, networks lack the ability to enable innovation, and second, they lack proper network abstractions. These limitations have resulted in an inability to keep pace with user requirements and to keep the costs of such networks under control. In part, this challenge to innovation stems from user requirements that are inflexible or unclear (e.g., user requirements that are stated informally, or user requirements that are statically bound to particular systems). Pursuit of the SDN thesis leads to the notion that control and maintenance of network infrastructure and services is better done by a machine which exploits the common control abstraction across all network devices, from programmatic configuration to monitoring and mechanized management.

We now describe examples where the use of SDN makes tangible impact on the complexity of a problem. The first shows how SDN can be used as an innovation enabler. In this case, SDN addresses an ongoing problem. A network operator wishes to try new ideas in an already complex network. The operator wants to understand what changes occur, how those changes have impacted the network, and to understand and interpret the resulting systems. Furthermore, in common with any good science, the operator wants repeatability with well-defined constants and variables—particularly in attempting to quantify the impact of the change. The idea of testing innovation within existing networks provided a core motivation of the original OpenFlow paper [5].

A second motivating example shows how a consolidated viewpoint impacts networking. A consolidated viewpoint is one whereby an observer could see and/or control an entire network rather than rely upon triggering a desired behavior by configuring many autonomous devices. A consolidated viewpoint also permits both improved behavior and new applications. Imagine a security scenario: a malicious machine is interrogating machines, probing for vulnerabilities, but doing so in a nondeterministic way. Observations of small amounts of malicious traffic may go unnoticed. However, centralized information would have identified the malicious intent faster through improved global awareness.

Now we consider a more sophisticated routing example. In a simplified network, all links are interconnected by routers, each making its own local forwarding decisions about the next hop to send a given packet based on destination address. In contrast, routing in such a non-SDN network is the result of a coordinated exchange of information about local connectivity, whereupon a routing mechanism can identify new or updated pathways. In a distributed network of routers, an operational failure leads to each router identifying the optimum path and making simple optimizations leading to local forwarding rules. However, despite a simple routing solution electing an apparent optimum, it is in fact a local minimum, and this solution can lead to overloading in (other) nonlocal links. The solution then involves each participant router iterating solutions in the hopeful (but nonguaranteed) pursuit of a global optimum. With more (nonlocal) routing information, a better global routing solution could be found, avoiding the intermediate local minima and improving convergence. A very simple example of this situation is shown in Fig. 2: Each host (H_A through H_D) communicates with host H_E, with a link capacity of 10. Initially, all the traffic goes through link L1. In a distributed network, if link L1 fails, each of the routers will autonomously try for the next best path. This means that initially all traffic will

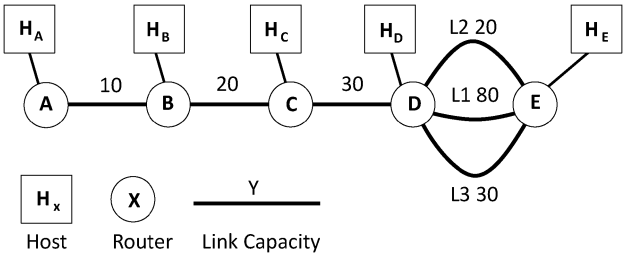

Fig. 2. *Simple network topology with multiple links between routers D and E.*

be directed to link L2. Suppose A and B succeed. Then C and D will fail, as the link will be overprovisioned. C and D will then autonomously try link L3 and succeed. In a network with centralized traffic management, such as provided over SDN, router D will first announce the failure to the traffic management application, which reprograms (through the controller) all routers.

As suggested by this example, the ability to make routing decisions is improved when a regional or global awareness of a system of routers is available. Without an SDN approach, the use of local-only information leads to poorer results in general and does not take advantage of well-established heuristics that can operate when knowledge and control on a broader scale is possible.

C. Actualization of SDN

To date, network complexity has been tackled by modular decomposition or modularization (breaking a problem into subtasks) and abstraction (dealing with ideas and roles rather than specific implementation details). The same principles apply to the use of SDN to limit network control complexity. SDN control functionality is commonly divided into multiple abstraction layers, in an effort to simplify and modularize the control tasks. Fig. 3 presents a generalized model of an SDN-enabled network control architecture. The architecture comprises three distinct layers: the data, control, and management planes.

The data plane, the lowest layer of the architecture, is composed of the hosts and devices of the network. In order to enable programmability by the control plane, the SDN paradigm builds a simple and clean functional separation of network devices, aiming to transform each device into a simplified forwarding engine that can be remotely controlled through a well-defined but restricted southbound interface (SI). The data-plane functionality of an SDN device comprises a limited set of operations, such as packet header parsing and extraction of a header field tuple, support for a fixed set of packet operations (such as header field manipulation and forwarding through a specific set of ports), and the ability to match packet header tuples against a lookup memory primitive (e.g., a hash table or a content addressable memory [27]). By contrast, the control interface of the switch enables an external entity to define the lookup memory entries and associate them with packet actions (e.g., forward any packet addressed to **A** using the *n*th port of the switch). Such control functions can also encompass the handling of exception packets, in cases where no specific handling rules exist for a packet, and for the accumulation of usage information such as packet counts. A final essential feature for the SI that shapes the SDN abstraction across layers is the flow-centric treatment of packets. In the context of SDN, a network flow is an ensemble of packets with header values that match specific ranges. For example, a TCP flow can be identified through a match with exact values for the IP addresses, the IP protocol field, and the TCP port numbers,

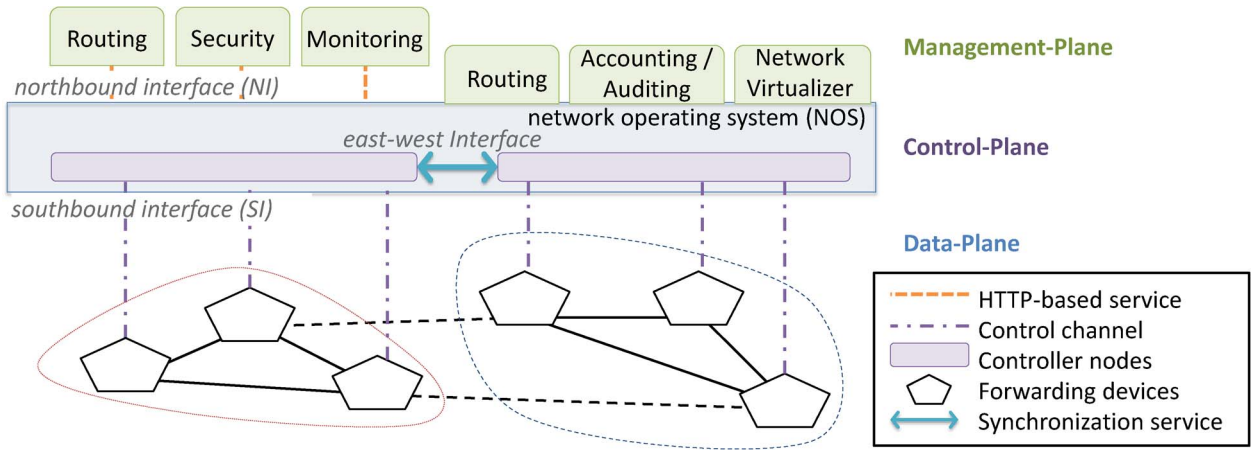

Fig. 3. *Model of an SDN control architecture. Functionality is separated in three layers: the data plane, the control plane, and the management plane. Integration between layers is realized through the southbound interface (SI), connecting network devices with the network operating system, and the northbound interface (NI), connection control application with the network operating system.*

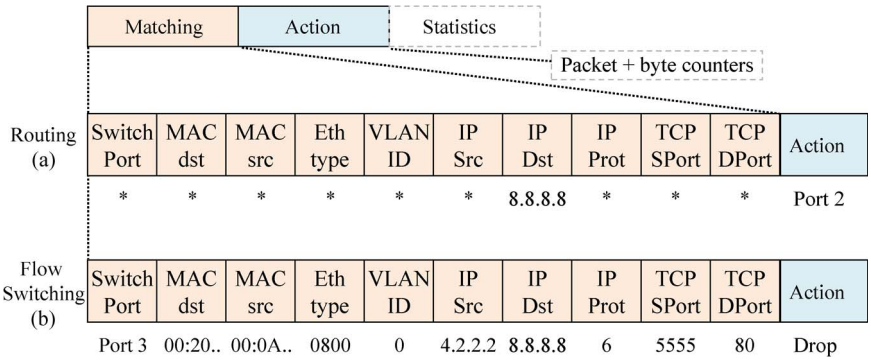

Fig. 4. ***Examples of matching rules and actions. The first example (a) shows a routing rule matching of a destination IP address 8.8.8.8 to output port 2. The second example (b) shows a rule matching each field in the header with a specific value, and dropping the packet if all the fields are matched.***

while a routing flow can be reflected through a match for the IP destination address. Effectively, the flow granularity is user defined and dynamic, comprising any set of packet header fields (Fig. 4 presents the available header fields in OpenFlow version 1.0). The flow abstraction is fundamental across all network devices and permits control convergence across different network elements (e.g., routers, switches, firewalls, and middleboxes).

In order to illustrate the design of a pragmatic SI, we elaborate the abstraction of the OpenFlow protocol. Each network device is modeled as a datapath, an ensemble of device ports and flow tables. The flow table is a core protocol abstraction reflecting the device forwarding policy.

Each flow table entry is split into three sections: the flow match, the action list, and the statistics. The flow match defines a flow using all the important header fields of a packet. Field wildcarding is supported, meaning any value will be accepted on the designated fields. The flow action list contains a list of packet operations, allowing header modifications and packet forwarding, applied to every matching packet. Finally, flow statistics of matched data include both byte counters and packet counters. Fig. 4 presents the structure of the flow table entry and two example entries. The first is a routing flow, matching only the destination address and forwarding packets with destination IP address 8.8.8.8 to port 2. The second is a flow switching entry, where all ten fields need to match the table entry. The action is set to drop a matched packet. We use these examples to highlight the generality of the OpenFlow abstraction to flexibly accommodate data-plane processing complexity. While the routing rule requires only a destination IP address extraction from each packet, the flow switching policy exhibits higher complexity. Because flow matches from different flow entries may overlap, the protocol assigns a flow priority to each flow entry in order to break ties. In addition, each flow entry contains optional timeout values, which identify the time period that a flow remains active in the flow table. Furthermore, the protocol provides message primitives to control the flow table, to query switch configuration and port and table statistics, and to intercept and inject data-plane traffic. The ability to intercept traffic is commonly used as an exception channel for traffic that is not handled by the device policy and, along with the network statistics polling messages establishes a set of powerful primitives to develop proactive and reactive control schemes.

The primary (operational) benefit of the SDN paradigm is the flexibility to rapidly develop new control logic in networking elements and effectively enable evolvability. In order to achieve this, the SDN paradigm exploits the flexibility of high-level languages and employs a control-plane layer to implement the network logic on general purpose servers. The control layer, often referred to as the network operating system (NOS), provides abstracted interfaces to network forwarding elements of varied capability and SI support, while managing contention for resources. Effectively, the control-plane layer is responsible for synthesizing the output of the control applications running on top of the NOS into a forwarding policy and distributing it to all the switches of the network. Additionally, the control plane is responsible for transforming input from the SI into semantically richer higher level abstractions, e.g., establishing the network topology. Furthermore, in order to improve the scalability and availability of the control plane, existing NOS borrow established techniques from the distributed systems domain to achieve horizontal scaling between control-plane nodes. Such distributed NOS employ an abstraction layer, the east–west interface, which allows seamless synchronization between the views of individual data-plane nodes. Such interfaces are implemented using distributed consensus protocols or popular database services, like the Infispan [28] distributed key/value store which is employed by the OpenDaylight controller [29]. While maintaining a common network view between distinct nodes, this approach permits the data-plane control requirements to be scaled among multiple servers.

There exists a tension between centralization and decentralization. Centralization is considered to offer enhanced network control and planning. For several of the SDN use cases (e.g., security or globally optimal routing), centralization is an enabling force. However, such centralization runs contrary to historic practice. Decentralized approaches are sometimes regarded as more resilient to failure and robust to changes in circumstance. While this tension is not resolved, significant efforts are taken by NOS (e.g., OpenDaylight) to ensure fundamental properties of resilience can be maintained by multiple redundant servers.

Finally, the top layer of the SDN architecture, the management plane, consists of the management applications which manifest the control logic of the network. This layer consists of common network control applications like firewall, access control and routing, but can also enrich the network capabilities by introducing new applications. Interaction between control applications on the management plane and the control plane is realized through the northbound interface (NI) of the NOS. The NI is defined by the control plane of the network. Its primitives vary between platforms, spanning from direct SI access to indirect access based on building new primitives (synthesizing multiple low-level interactions of the SI). Control applications can run on hosts separate from the control-plane nodes, accessing the NI through various standard services, or they can be physically integrated with the control layer during the compilation of the NOS.

III. PROGRAMMING LANGUAGES AND PLATFORMS FOR SOFTWARE-DEFINED NETWORK RECONFIGURATIONS

The SDN paradigm defines an abstract architectural model for the control of the network and identifies some key design properties (e.g., flow-centric treatment of traffic). The realization of SDN across the different layers of the network still remains an open question for the research community, which we elaborate on in this section.

The majority of research efforts in the field of SDN programming languages and platforms have focused on the control and management planes, while data-plane research, with its line-speed expectations, has limited research to only a few facets based upon reconfigurable devices.

A. Management Plane, Control Plane, and NI Programming Languages

The management and control plane of the SDN architecture orchestrates the network logic. The primary design goal of the control plane is to expose an application programming interface (API), the NI, which allows developers to focus on programming the network rather than the device, thus abstracting the effort to consolidate control across multiple devices and locations. In addition, control-plane platforms aim to construct new reconfiguration abstractions, by synthesizing low-level reconfiguration capabilities of the underlying network infrastructure. For example, heterogeneous support of SDN reconfiguration capabilities across the network forwarding devices (e.g., variable support for fast-path packet modifications between forwarding elements) can be abstracted through runtime network policy optimization (e.g., setting up end-to-end paths that apply packet modifications on the most appropriate device of the path).

The NOS is the main building block of the SDN control plane. Similar to a traditional operating system, it executes management-plane applications, and it is responsible for coordinating access and securing network resources. Early NOS approaches, like Ethane [30] and NOX [31], provided low-level OpenFlow protocol translation and multiplexing and supported basic network services, like switching Ethernet address monitoring in order to map them to network device ports and minimize traffic broadcasting) and user-based access control (strong user authentication and association with a network policy that allows use of specific applications). Nonetheless, the wide adoption of the SDN paradigm has motivated the enhancement of NOS with novel capabilities, like NOS scalability through state sharing between different instances, monitoring, policy conflict detection, and resolution between management applications and network virtualization. As the SDN paradigm is deployed in production networks, an interest is put toward mature control-plane platforms, supporting a richer set of network service. As a result, a series of vendor and service provider consortia have been formed currently in an effort to develop and support NOS platforms (like the OpenDaylight [29], ONOS [32], Ryu [33], and Floodlight [34] platforms). At the moment of writing, the standardization of the NI of the NOS still remains an open question. The abstraction varies between existing NOS, and it is highly influenced by the target deployment environment (e.g., a controller targeting carrier grade networks, like ONOS, requires a different set of control-plane functionalities, in comparison to a controller targeting virtualized data centers, like the VMware NSX [35]). Although a detailed discussion of the NI is beyond the scope of this paper, it is interesting to note that as the NI becomes more tightly coupled with the underlying controller function (e.g., routing versus security), its semantics will tend to converge to a common definition [3].

The development of the data-plane layer has created an interest in effective management-plane development environments using domain-specific languages (DSLs). Management-plane DSLs are built on top of the control plane and use the NI provided by the control plane. The novelty and effectiveness of the SDN approach has developed an interesting competition between programming language experts to define new DSLs with support for all the required programming primitives and semantics. This led to the development of multiple languages aiming to address different aspects of control-plane programming. For example, Netcore [36] provides a high-level forwarding policy language, Nettle [37] transforms the view of

control applications and focuses on changes in the state of network elements rather than event processing, while Maple [38] provides a scalable multicore scheduler and runtime policy optimizer for OpenFlow control to match available device resource configuration. Nonetheless, several programming languages have been developed by evolving existing languages, such as Flog [39], which combines ideas from FML [40] and Frenetic [41]. Research on SDN DSLs has explored also the applicability of different programming paradigm on the expressability of the management plane. With the exception of Pyretic [42], most high-level SDN programming languages adopt a declarative paradigm, then explore further specific programming models: functional programming [36], [38], [41], logic programming [39], [43], dataflow programming [40], and functional reactive programming [37], [44]. At the time of writing, no single language stands up to all the challenges imposed by SDN. We do not expect this situation to change in the future.

Research on management-plane applications has motivated solutions for a wide range of network problem. To exemplify how these solutions are leveraged through the SDN paradigm, we will focus on a common network problem; maintaining consistency during policy updates. The centralized nature of SDN introduces a significant problem in incremental policy update deployment. A policy update for a network path that spans across multiple switches can result in transient policy violations, if the processing of the flow table modification messages is not timely and ordered across all switches. Nonetheless, the semantics of the reconfiguration abstraction in existing commercial off the shelf (COTS) platforms is not designed to provide such update consistency semantics [45]. Traditional network control protocols support weak consistency models, using distributed eventual-consistent algorithms (e.g., routing protocols) [46]. In the context of SDN, multiple solutions have been proposed to address this problem, by introducing in the NOS NI transactional update interfaces. The NOS implementation of these interfaces aggregate policy updates, analyzes them for potential conflict during deployment, and schedules their deployment accordingly using two-phase commit algorithms [47]–[49].

B. SI Programming Languages

Existing SDN approaches have widely adopted the OpenFlow protocol [5] as the SI. Released as an open-source implementation that fulfilled a need, OpenFlow has become widely available in commercial SDN devices. OpenFlow holds an important position in its role as an early SDN enabler. It assimilates a low-level, assembly-like machine language, closely aligned with and limited by the underlying hardware. The burden remains on the programmer, who needs intimate understanding of the hardware (such as switch details and available resources), as well as behavioral details of the handling of overlapping rules and rule ordering. This was one of the incentives for the development of the management layer languages. Nonetheless, such limitations reduce OpenFlow code reusability. Consequently, creating modular/reusable code is challenging and the development process is prone to error.

At the time of writing, OpenFlow does not have sufficient expressability to cover the entire functionality provided by network devices, nor can it optimize their performance. This is true even for devices having an architecture compliant with the OpenFlow specification. For this reason, another abstraction layer is often provided between the two. For example, Broadcom's OpenFlow data-plane abstraction (OF–DPA) [50] defines and implements a hardware abstraction layer that maps the Broadcom's StrataXGS switch architecture to the OpenFlow 1.3.1 switch and pipeline. Similar approaches are currently explored by other vendors. The protocol oblivious forwarding (POF) [51] proposal sets an ambitious goal to provide an abstraction table above the device driver, but also to extend OpenFlow's protocol-dependent instruction to be protocol independent. A prototype POF implementation is available for the Huawei's NE5000 core router, which uses the proprietary microcode for its network processor. Furthermore, P4 (an acronym for programming protocol independent packet processors) [52], [53] sets three more ambitious goals: switch reconfigurability in the field, protocol independence, and independence from underlying hardware. In this way, P4 operates as a complement to SDN protocols, like OpenFlow, and considers reconfiguration of the data plane to support (in target hardware or evaluation designs) specific operations that are then manipulated by such protocols as OpenFlow.

C. Data-Plane Programming Languages

While OpenFlow handles the low-level aspects of the data plane, it is not the language used to program widely used data-plane devices. Network processors, whose host processing units require special programs, were for many years using proprietary programming languages (e.g., Marvell's XLP [54], EZchip [55]). The use of specialized processing units, optimized for bandwidth, led to the development of different instruction set architectures by each company, exploiting the advantages of each architecture. In 2008, Cisco was the first company to introduce a network processor that was fully ANSI-C compatible [56]. This approach was later followed by Ericsson [57], and most recently by EZchip [58].

Several attempts have been made to go beyond assembler and C-like programming languages for packet processing. PacLang [59] was an early attempt for a high-level data-plane language prototyped on the Intel IXP2400 network processor. Based around the premise of strong typing (explicit variable type casting, checked at compile time) and linearizable types (any object variable is used exactly once within the program, thus simplifying memory management), PacLang presented a transformation-based methodology to separate architecture details from the

high-level program specification. The application code was written in a high-level language and then matched to the network processor architecture using an architecture map, ping script. Compared to previous solutions for a single task or pipeline, a novelty of PacLang was its ability to handle multicore network processors.

PX [60] (and its earlier incarnation [61]) is a high-level language for specifying packet-processing requirements, designed for field-programmable gate array (FPGA) implementations. It is also focused on what should be done, rather than how, leaving the hardware implementation details to be handled by the compiler. The compiler, in turn, generates code in VHDL and Verilog hardware design languages (HDL).

While no implementation is available to date for network applications, functional languages also present a future direction for reconfigurable network devices. Languages such as Chisel [62] and HardCaml [63], which generate low-level Verilog or VHDL code, are suited for such purposes.

D. Proprietary Environments

While the predominant SDN realizations remain currently under the umbrella of the Open Networking Foundation (ONF) [64] or IETF [65], some organizations choose to have their own environments supporting the same concepts. This allows these organizations to maintain a proprietary environment, implementing mechanisms that best suit their hardware and software, easing customer migration and removing restrictions imposed by public standards or specifications.

Cisco's open network environment (ONE) tries to go beyond SDN and set the foundations for, what Cisco describes as, an Internet of Everything (IoE) [66]. Their approach does not reject SDN, but rather tries to extend it to create a better integrated solution for Cisco devices. Accordingly, Cisco's network processors (e.g., nPower X1 [67], Typhoon [68], and QFP [69]) are OpenFlow capable. The difference lies in the development environment underlying it, dubbed onePK [70]. onePK allows a programmer to write code in one of several languages (C, Java, Python) using a set of APIs that abstract the OS and network device internals. Effectively, onePK enables easy interoperability with multiple layers of the SDN model, as well as other interfaces and languages, like OpenFlow and HTTP-based services [71], either seamlessly or through plugins [72]. The onePK environment also integrates with Cisco's application-centric infrastructure (ACI), which operates at a higher architectural level.

A very different approach is taken by Xilinx's software-defined specification environment for networking (dubbed SDNet) [73]. SDNet assumes that the underlying hardware is completely programmable (e.g., FPGA), and uses this to implement programmability of the data plane. The concept contains a complete design flow, from SDNet high-level description language, through the SDNet hardware design language compiler, to Xilinx's design tool (Vivado) that generates the FPGA implementation's bitstream. The data-plane packet processing units allow firmware updates between packets. While SDNet is not tied to a specific southbound programming language, it does not reject them either: the user may choose to implement, for example, OpenFlow protocol support in hardware, and provide custom code to support it.

Additional environments, such as Juniper's Junos Fusion, Huawei's SoftCom, and Arista's software-driven cloud networking (SDCN), exist, with various levels of maturity and conformance with ONF. The adoption of these (primarily commodity/closed-source) environments by the networking community is yet to be seen.

IV. RECONFIGURATION IN ELECTRONIC SWITCHING

In current electronic switching, software-defined networks commonly rely on header processing. Header processing is the stage where the header (the part of the packet that contains address and network-handling details) is identified and examined. This is also the stage where packet actions are decided, such as setting a packet's destination within the device (e.g., queue, flow, port), or selecting the number of replications of a packet. However, there are further reconfiguration aspects to switching, which are discussed in this section.

A. Header Processing

A networking element does not always require header processing. For example, a host computer attached to the network with a single-port network interface card (NIC) may forward all packets from the network to the central processing unit (CPU), and all packets from the CPU to the network, without further processing. However, this is rarely the case. Recall simple operations of our basic switch in Section II: as a packet enters a device, its header is parsed, matching rules are checked, and actions are applied to it. The analysis may examine specific bits or detect an expected format (Fig. 4), or perform a more sophisticated parsing. Clearly, the level of reconfiguration required for this stage may vary between devices. Devices that support only one header type may have little need for reconfiguration, but devices supporting more sophisticated header processing might benefit from the ability to dynamically reconfigure hardware to perform functions more effectively.

Highly reconfigurable systems could permit defining which protocols are admissible (or excluded), the number of headers within a packet to be looked up, or even describe complex lookup operations when multiple networking protocols are being used. Furthermore, reconfiguration flexibility allows adding support for new protocols. As new protocols emerge, a network operator may need devices to recognize these protocols and handle the packets accordingly.

It is instructive to consider a few examples of how network actions can be driven by the structure of packet headers. For example, a most basic admission action is "Should the packet be admitted or dropped?". Other example actions are to assign a packet to a specific destination output port, or to specify a quality-of-service queue within the device. Packets terminating within a device (e.g., exception packets) may have actions that differ from those for packets sent to a remote destination. The stage at which actions are assigned can also vary, yet three stages are commonly localized: The classification stage (where necessary information is extracted from the header, such as the protocol, the packet's source, and destination), the forwarding stage (where the destination queue and/or output port of the packet inside the device is decided), and the modification stage (where the header of the packet is being altered). The shared property of all these stages is that they require configuration, both of actions and of results.

The header processing module may conduct many further operations, from collecting statistics to security operations (e.g., dropping packets with a false source address). Such functions can be rich, varied, and almost arbitrarily sophisticated, attesting to a need for expressive forms of SDN.

Currently, the hardware involved in network systems support a variety of reconfiguration mechanisms. The simplest involves using registers for configuration, yet their use tends to be limited to enabling or disabling a function. A standard header processing configuration is implemented using tables or databases. The most common table is a forwarding table. The routing table is an example of dynamically configurable data structure associated with the routine operation of a router. During operation the router learns new IP addresses, along with a port assignment through which they are accessible. This information is added to the routing table and new packets arriving to the router are sent to the right port accordingly. As network connectivity changes over time, so does the routing table: entries are not only added, but also deleted or altered. While the routing table can be considered a stored state, the result of modifying the routing table is an operational reconfiguration: packet flows previously sent to port X may be sent to port Y as a result of an entry being modified, their content (e.g., header fields) may be modified, or e.g., they may be dropped.

Header parsing uses tables that are indexed by packet headers, e.g., destination IP addresses. Each entry in such tables contains specific primitive actions such as setting the packet's destination, drop the packet, etc. These tables used for header parsing require memory, which in turn scales in direct proportion to the number of entries in a table. For this reason, for many years these tables were implemented using external memory modules, using different memory technologies. Over time, shrinking silicon processes allow more and more on-chip memory. As external memories not only add to a networking element's cost, but also to its power consumption and overall size, there is a considerable motivation for using on-chip memory to implement these tables. This trend in monolithic implementation stands in contrast with the progressive need to increase table size to accommodate more table entries. Consequently, some network and packet processing devices use external memories, while other contain all tables on-chip. A variant of those are configurable networking devices that allow selection between relatively small internal and larger external memories.

Network and packet processing devices which employ many tables and allow a large flexibility often face the challenge of meeting conflicting size requirements by different customers: one application will require table A to be large and table B to be small, whereas a second application will require a lot of entries in table B and no use of table A at all. This contradiction can be solved by sharing databases across a device: allowing a user to select the memory size for each table out of a shared pool. This approach is resonant to ones often used in FPGA devices, where the FPGA provides users with a shared pool of memory resources that can be utilized according to an implementation's needs.

The flexibility expected in header processing has grown over time. If two decades ago a static configuration was acceptable, and a decade ago marked the emergence of network processors, then today many devices claim to be fully programmable and highly flexible. This is largely driven by market forces, as chip vendors try to reach as many market segments as possible. In addition, users require programmability in order to be able to reconfigure their network over time, adding new protocols, altering configurations, and so on. While in the past network processors used proprietary processor architectures that maximize performance (e.g., Marvell's DataFlow architecture [74]), then today more and more network processors embed "traditional" RISC architectures (such as EZchip, Broadcom, Ericsson, Cisco). This trend is possibly another step toward the less intelligent programmable hardware driven by the SDN paradigm.

B. Traffic Management

Header processing is focused on where packets are sent. In contrast, traffic management is focused on how a stream of packets to a certain destination is being handled, which is commonly referred to as the quality of service (though traffic management is broader than that). Quality of service covers many parameters (such as bandwidth, latency, and jitter) and is provisioned using different types of mechanisms within a traffic management device (e.g., scheduling and rate limiting). As traffic management devices need to match the services bought by the user to the available resources, they tend to be highly configurable. Such devices require the ability to intimately configure and tweak resources, allowing every traffic flow to be assigned to a correct group of detailed servicing rules.

We distinguish between two classes of reconfiguration mechanisms: mechanisms that affect the way the device works, and mechanisms that set the way a specific traffic flow is handled within the device. The second type of configuration mechanism is typically easier to handle, since it is usually implemented within a table, written as part of the power up sequence. The entries in such a table indicate what different properties should be assigned per flow. Examples include setting the level of priority or the committed bandwidth.

Configuration mechanisms of the first type (that affect intrinsically how a device works) can vary significantly. For example, a scheduler may allow one or more scheduling schemes to be defined (such as strict priory versus weighted fair queuing [75], [76]), and weights need to be assigned to any flow or group of flows in a weighted scheduling scheme. Similarly, a traffic shaper needs to be assigned average and peak rates as well as a burst size [77].

Congestion management is another class of complex network operation amenable to (if not requiring) reconfiguration support. Example functions range from the simple setting of thresholds in different queue management schemes (such as random early detection [78], where a packet is dropped before being admitted to a filling queue) to rate adaptation (by methods such as explicit congestion notification [79] and quantized congestion notification [80]). Changing the traffic rate of a flow requires changing the configuration of thresholds and shaping parameters in a networking device, which in turn modifies the rate of a given flow (e.g., a congested flow).

Despite being a general paradigm, when OpenFlow was introduced and SDN got traction, the focus of data-plane research was on header processing. There is a growing understanding that other aspects of the data plane need to be defined as well. Furthermore, as the central management of SDN allows an end-to-end view of resource utilization across the network, using it to improve aspects of traffic management is called for. The use of the control plane for traffic management was well studied by different groups (as surveyed in [3] and [81]). Implementations of traffic management in the data plane, in hardware (as opposed to software-based solutions, such as QueuePusher [82]), are still rare. One example of such implementation was presented by Sivaraman *et al.* [83], who implemented SDN-enabled queue management in an FPGA. Further study of enabling traffic management in the data plane is still underway [84]. The lack of research done to date in this area is somewhat surprising, given that it was proposed as a characteristic of SDN a long time ago [1].

C. Switching Devices and Functions

The last building block of a switching device that we discuss is the switching unit. The most basic switching unit is the crossbar, which allows a dynamic connection between input/output (I/O) pairs between the ports of this network element. For this switching method to be nonblocking (meaning all possible combinations of I/O pair assignments can be accommodated), it must be configurable, allowing inputs to change their paired outputs over time, e.g., allowing packets incoming on port N to be sent to any port M, according to their header. This type of switching is near instantaneous in electrical switching but can take milliseconds in some electromechanical photonic switches (discussed in Section VI-B). As crossbars (whose internal resource consumption grows quadratically with port size) fail to scale with the performance required in modern networks, other, more scalable, multistage switching architectures are gaining traction within current day systems (e.g., Clos [85] and Fat-Tree [86]).

Switching elements have several modes of use. For example, a network switching chip can work as a standalone device (with all its interfaces serving as ports), or it may be connected in a mesh with other identical devices (to create a system capable of higher radix, bandwidth or both). In less common cases, a device may connect to a larger fabric mesh to create a multiboard or a multichassis switch (e.g., using commercial devices such as Broadcom's BCM88750 and BCM88650 [87]). This is commonly achieved using modular (i.e., board or box) assemblies. In this case, the end user buys a module, which can be used in a variety of different ways. A fabric module can be configured to operate as a single-stage switch fabric in a standalone chassis, or as a first and last stage (but, for example, not middle stage) fabric switch, connecting to a different fabric chassis when placed in a multichassis system [88]. This type of a configuration allows scaling switching systems based around common building blocks, such as Huawei's NE5000E and Cisco's CRS-X, from a few terabits per second to hundreds of terabits per second [88]–[90].

Networking devices regularly offer more programmability and reconfiguration than available to the end users. Decisions taken during the design of a system, and settings applied during the assembly of devices within these systems, limit the level of reconfiguration available to the end user. Consider the example above, where devices used to create a multiboard or a multichassis may have programmable modes of operation. These devices also typically support multiple types of physical interfaces (e.g., 10 GbE, 40 GbE, 100 GbE), but once assembled on a given board the interface type is set to match the optical transceiver of this module and cannot be altered. This means the end user can alter the device or module's role within a system, but cannot alter the type of physical connectivity. This start-of-life setting benefits both silicon vendors and their customers: silicon vendors design and fabricate only one chip to support different market segments, whereas their customers use the device's programmability to manufacture and sell the most power-efficient, cost-effective networking system.

At the time of this writing, SDN-enabled electronic switching devices are increasingly being introduced by commercial vendors. This trend is also emerging in the latest photonic switches discussed in Section VI.

V. TECHNOLOGIES FOR RECONFIGURABLE NETWORK SYSTEMS

Several types of technologies are applicable for networking devices. In considering how reconfigurable systems approach could be used more effectively, we consider a spectrum that trades configurability for performance, and what we might do to better achieve both. This section provides an overview of these considerations and discusses the trends over the last few decades.

Let us first describe the spectrum of programmability and configurability. The most programmable type of device is a general purpose CPU. Completely programmable, it allows any programmer to create a networking device of his own design, where only the device interfaces set the limitations of the CPU-based system.

Network processors (we omit graphics processing units from this discussion) are less programmable than CPUs, as they are designed for a specific purpose, and their architecture matches that purpose. Network processors vary considerably in their architecture, therefore it is hard to make generalizations about their level of programmability: some network processors force a single datapath structure and allow programming the actions taken in every stage of the datapath, whereas others allow flexibility in the structure the datapath itself. The level of expertise required to program a network processor is higher than required to program a CPU, as the programmer often needs to write (device-specific) programs in order to configure each processing unit within the network processor.

We consider traditional FPGAs to be less programmable than network processors. An FPGA device is built from a set of resources (programmable logic blocks, memory blocks, and I/O). Using HDL (or high-level languages generating HDL descriptions) a user can design the FPGA to perform any operation, limited only by the available resources. Once completed, the design is then downloaded to the FPGA device. Once a resource is configured to work in a certain way, it will maintain this function until the device is powered off or the device is reprogrammed. This limitation on the use of a resource within the FPGA makes the FPGA less programmable than a network processor.

FPGA vendors offer processing cores embedded within the FPGA (e.g., [91] and [92]). These processing cores can be either soft cores (built using the FPGA's general purpose logic), or hard cores (built from dedicated silicon) [93]. Hard core processors (e.g., [94]) typically offer a better performance than softcore processors (e.g., [95] and [96]), however their dedicated silicon presents a waste of resources for FPGA designs that do not require a processor. Implementing a networking device over an FPGA requires a larger set of skills than programming a network processor. Even when using high-level programming languages, the user needs a deeper understanding of hardware aspects. The implementation of an FPGA (including simulation, synthesis, and routing) requires a set of skills not traditionally possessed by software engineers.

The least configurable devices are application-specific integrated circuits (ASICs) and COTS networking devices. While the level of reconfiguration of these devices varies, from highly configurable to completely transparent, as a group they are far less configurable than other solutions. COTS devices provide a closed feature set that the user may choose if and how to use, but the function itself is rarely programmable. Some contemporary high-end devices introduce a level of programmability into their packet processing units, using processing cores, however we classify those as a hybrid with microprocessors. While using a COTS device often requires the least expertise, designing one is the most challenging. It engages people with a wide set of expertise, from frontend and backend designers to embedded software engineers. Note the distinction that we make here between CPU and COTS networking devices: CPUs share with COTS devices a very long and expensive design cycle, but once CPUs reach the market, the customer can use them for a wide range of applications. A COTS networking device has its application set during the design stage, and it cannot be changed after that. This also reflects on the level of risk: if a bug is found in a networking device implemented over a CPU, the effort required to fix the bug is minimal. Fixing a bug in a COTS device can have an indeterminate complexity, often requiring changes in the manufacturing masks or even a new device fabrication, costing millions of dollars.

A. Evolution of Networking Devices Bandwidth

While CPUs offer the best level of programmability, they are not present in all networking devices. High levels of programmability are mostly seen in slower devices, while the highest bandwidth is commonly provided by the least configurable devices. Economic drives propose one reason for this phenomenon, but a technological insight is provided in Fig. 5. This figure presents the evolution of aggregated bandwidth of networking devices over several decades, starting from 1 Gb/s of the early 1990s to contemporary performance levels exceeding 1 Tb/s. As the figure shows, from the 1990s, CPUs (the most programmable devices) provided the best performance. Over time COTS devices (the least reconfigurable) managed to deliver the highest bandwidth. To better understand this conclusion, we explain the details of the figure.

For network processors and COTS networking devices, we use the manufacturer's declaration of bandwidth. For FPGAs, we present the maximal theoretical bandwidth, calculated using the IEEE 802.3 standards methodology where the quantity of I/O that can be used by a networking interface is multiplied by its maximal frequency. The definition of bandwidth per CPU is the product of device's bus width and core frequency, for a single core. Each point in the graph presents the "best of breed" performance for a given device type at a particular snapshot in time. All known devices available on the market to date were considered, including but not limited to Intel, AMD,

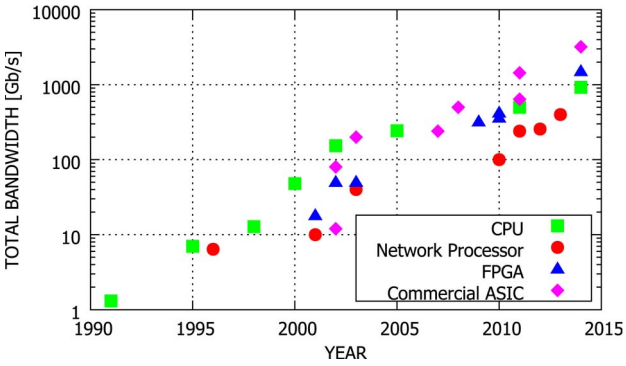

Fig. 5. ***Evolution of networking device bandwidth. In the last decade, the least programmable devices (COTS) provided the highest bandwidth performance, whereas the most programmable devices (CPU) made the least improvement.***

EZchip, Netronome, Broadcom, Marvell, Mellanox, Altera, and Xilinx.

For CPUs, we calculate the effective bandwidth as the product of bus width and core frequency, as it is an indication of the internal pipe bandwidth as well as the possible packet rate (the number of packets processed every second). Although multicore processing was the direction taken to improve CPU performance, the product of the above bandwidth and the number of cores is not equal to the total device bandwidth. The internal communication bus of multicore CPUs is not designed to match the core bandwidth, even given optimizations [97]. As the multicore communication is the primary bottleneck, and its throughput is less than a single core's, we present a single core bandwidth and consider intercore communication a challenge for future networking initiatives, such as the Intel Omni-Path fabric [98].

Fig. 5 also reveals that in the previous century, CPU performance led all other solutions, while the last decade shows a more modest improvement of datapath bandwidth performance per core. In a comparison of 2007 and mid-2015 best-in-breed CPUs across all four SPEC CPU2006 benchmarks [99] we make an important performance observation. For whole-system performance, the speed benchmarks (CINT2006 and CFP2006) improved by a factor of approximately four and the throughput benchmarks (CINT2006rate and CFP2006rate) by a factor of 15, yet the relative performance per core improved by less than two, and the relative throughput performance per core improved by less than a factor of four. This difference also contributed to the emergence of network processors.

ASIC/COTS devices have played an interesting role in the evolution of network devices. A single modern network device may consist of many types of high end silicon devices, e.g., traffic managers (implementing packet rate limiters such as described earlier) or packet processors (devices implementing header parsing and header-field lookup to trigger specific processing). Yet the highest bandwidth capacity is always presented by the switching devices (those devices that, having the highest port count, provide port-to-port connectivity). The lower the complexity of the device, the fewer are the features beyond just switching, and the higher its bandwidth. The complexity of the device impacts both silicon area and difficulty in achieving a design that meets all the criteria. This drives devices toward simplicity, reducing features, in order to improve performance. This optimization of the ASIC and COTS devices is evident in comparison with network processors, which are built in the same semiconductor processes and subject to the same manufacturing constraints as CPUs, FPGAs, and other devices. It should be noted that the growth of bandwidth in COTS devices is tightly related to the mechanical aspects of the target switching systems. The bandwidth is often a multiplication of 24 ports times the maximal interface speed available, as this is the number of ports fitting a front panel of a card in a 19-in rack. Thus, 24, 48, and 96 ports are commonly used for port count, with the addition of a few uplink and management ports.

FPGAs have shown a consistent improvement over the last decade, with a steep increases in performance. On the other hand, the graph also demonstrates why FPGAs do not gain more traction with system vendors: the place where the FPGAs match the COTS devices performance is always at the same time or after COTS devices became available on the market. This means that the FPGA-based design will only start when a COTS-based design is already available. We account for early access to FPGA technology by large vendors and the reduced design cycle time of FPGA. However, eventually the networking devices need to be tested and validated in the lab, which can take a considerable amount of time. In order to replace COTS devices, FPGA devices need to be available a couple of years ahead of COTS devices with the same clock frequency (and other performance characteristics). Altera's Stratix 10 is an example of such an attempt, as Altera gained early access to manufacture it in 14 nm by an Intel semiconductor fabrication plant [100].

The design of a networking device is more than just interface bandwidth and clock frequency. Other FPGA resources (e.g., logic elements, embedded block memories, etc.) need to match those of COTS networking devices. The FPGA must provide a superset of all these resources or the design will not be able to carry the required feature set. This raises the complexity and cost requirements of FPGAs. Having said this, we prefer not to engage in a direct comparison of FPGA to COTS devices as we think a fair comparison is difficult. This is primarily due to the fact that simple metrics (such as gate count) do not adequately capture the true capabilities of an FPGA.

B. High-Speed Interface Adoption By Networking Devices

Fig. 6 presents the adoption of networking interface by different classes of networking devices over time. In this

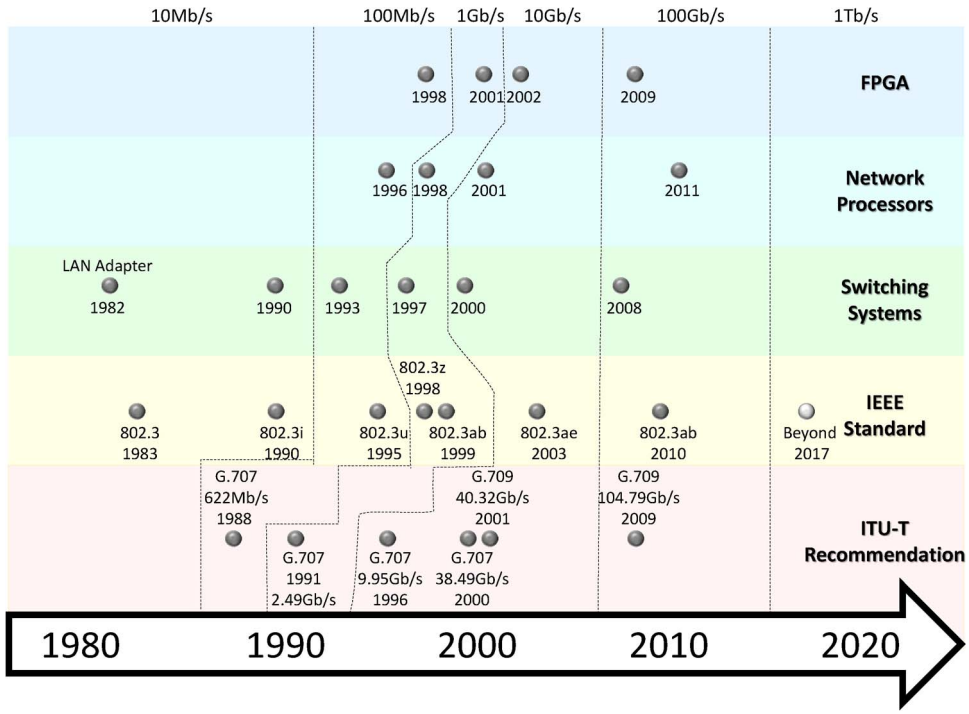

Fig. 6. ***High-speed interfaces adoption by networking devices. Commercial switching systems tend to adopt new high-speed interfaces before a standard is concluded. FPGA devices do not claim support of new interfaces before a standard is completed.***

figure, we focus on members of the IEEE 802.3, ITU-T G.707/Y.1322, and ITU-T G.709/Y.1331 standards families, starting with the basic 10-Mb/s IEEE 802.3 standard, and spanning the higher speeds projected by (at the time of writing) future standards. Interfaces composed from several parallel lanes, such as 4 × 3.125 Gb/s XAUI, are considered by their aggregated bandwidth (e.g., 10 Gb/s). The figure shows over time (*x*-axis) when a particular technology class achieved a particular performance plateau. The introduction of the new IEEE and ITU-T standards is shown by the two bottom rows. A particular point on the graph indicates an approximate point in time of "initial adoption," when products that embodied that interface performance level were available from the designated class ("FPGA," "network processors," "switching systems"). We choose to refer here to "switching systems" rather than ASIC/COTS devices, as most COTS switching devices do not connect directly to the network, but instead are interconnected with a vendor-specific (non-IEEE 802.3x or ITU-T G.709/Y.1331) interface, such as Interlaken [101] or SPI4.2 [102]. As such, the vendor-specific solutions more readily achieve targeted performance levels between their own components using proprietary ASICs.

While one would have expected the networking systems to be the last to come into the market, as their design cycle is unlikely to exceed the availability of silicon devices, the contrary thing happens: since the mid-1990s, commercial systems always came into the market ahead of the official announcement of standards. While this is driven by companies wishing to gain customers by being first-to-market, this also relates to the long life-cycle defining standards and the involvement of vendors in this process. This involvement allows them to complete their design ahead of the publication and wide availability of the standard. The introduction of new standards varies significantly between standardization bodies. As Fig. 6 shows, ITU-T recommendations were once ahead of IEEE by up to seven years. While we cannot attest to the reason, we believe it is the combination of the focus of ITU-T on optical communication rather than electrical, and the narrow market segment aimed by the G.707 and G.709 standards (i.e., optical transport networks), which allowed fewer requirements and faster consolidation.

Network processors appear also to follow the same trend: being early in the market, available to the system designers that require them. An anomaly is observed for the introduction of network processors supporting 100-Gb/s interfaces, announced later than the standards. This is possible as several network processing devices announced in 2010 (the year of the IEEE 802.3ba standard) were not able to reach an aggregated 100-Gb/s bandwidth, and the one that did (EZchip NP-4) used an Interlaken interface (an interconnect protocol driven by Cisco and Cortina Systems).

The desire of FPGA vendors to gain a larger market share is demonstrated by their adoption of high-speed interfaces. While in the past FPGAs were last to adopt new

standards (e.g., 10 Gb/s), today we see a trend for FPGA transceivers to match or exceed the developing standard. For example (at the time of this writing), Altera is announcing in its newest FPGA devices (Stratix 10) transceivers capable of 56 Gb/s, ahead of the 25.7 Gb/s used by 100 Gb/s CAUI-4. This appears aimed at the optical internetworking forum (OIF) 400-Gb/s standard [103]. The advantage of FPGAs in this field is that once their transceivers meet the electrical requirements of the new standard, the FPGA companies can design and release logical cores (e.g., PCS–PMA for the physical coding and media-attachment layers, and new MAC protocols for the data-link layer) at any point in time without affecting the customers hardware. Such processes were easier when all standards used a two voltage level, e.g., non-return-to-zero (NRZ) transmission, but with new proposals involving more sophisticated physical layer interfaces, including multilevel coding, e.g., pulse amplitude modulation (PAM), support of a standard cannot practically be announced before all transceiver requirements have been settled. As if to demonstrate this point, Xilinx announced support of 100 Gb/s in 2008 in the Virtex-5, though its transceivers maximum rate was 6.25 Gb/s [104]. Xilinx has also announced support for 400 Gb/s [105], demonstrating a 400-Gb/s MAC capability using an Interlaken interface [101] as (at the time of this publication) no 400-Gb/s IEEE or ITU-T standard has been ratified.

C. Power Consumption

Efficient power consumption is a common challenge across all types of networking devices and networking markets. In the high-end market, the power density of silicon has grown to the point where devices limit their feature set in order to meet the power budget. Configurability is one of the main mechanisms today to address the power budget limitations without reducing any device feature set [106]. A silicon vendor or an FPGA designer can design a device which addresses requirements for a wide market segment and provides configuration options that reduce power consumption when a feature is not required. This may be implemented as configurable table size, powering on and off of certain stages/units/blocks in the design or configuring the device frequency (and thus its bandwidth). Many more power saving by configuration techniques exist, which are not limited to networking devices (e.g., voltage scaling). New manufacturing processes promise considerable power saving, yet none of these leads to a ground breaking reduction in power consumption. Photonic switches (Section VI-B) can provide power saving, however, they remain applicable only to fabric switching devices and not, for example, to network processors.

D. Balancing the Forces

The networking market is not altruistic, and the eventual goal of all system vendors is to sell more products. To this end, each vendor wants to be first in the market, presenting the most advanced solution with the richest features set, the highest bandwidth, and the lowest power consumption. Unfortunately, life is a series of compromises, and designing a networking device is no different. We have already mentioned time to market and compliance with standards, but this is not the only limitation. Silicon vendors often find themselves limited by technology constraints, more so than architecture limitations, e.g., the maximal power or power density of their device as well as the maximal silicon area with a reasonable yield. By today's standards, the scale of integration of a typical network ASIC is remarkable: Intel's Xeon E7 v2 employs 4.3 billion transistors for a die area exceeding 500 mm^2 [107]. Xilinx's Virtex-7 2000T FPGA device is built from 6.8 billion transistors [108]. Over seven billion transistors are floorplanned in devices such as Broadcom's Tomahawk [109]. Factors including integration scale, performance demand, and competitive pressure drive a complex set of interrelated decisions that face the design of any new device: should the number of lookup entries in a table come at the expense of a certain protocol support? Should multiple interface standards be supported at the expense of area or latency? As silicon vendors try to address a large number of customers, in-house ASIC designs can often reach better capabilities, as they only need to suit their own system needs: the size of lookup tables and supported protocols are set by their end system, and its front panel options define the interfaces that will be supported. This reduces the overall number of features, but improves the quality of each.

VI. PHOTONICS

As data rates have risen, copper cables have increasingly been replaced by optical fiber. For point-to-point links, photonic communication offers clear advantages in lower energy consumption per bit and increased signal integrity at high bit rates and long reach without the use of digital signal processing (DSP) [110]. The increased reach and signal integrity also provide latency benefits due to reduced buffering and DSP delays. In contrast, packet switching (switching on a per packet basis) has remained in the electronic domain. The only area of widespread commercial adoption of photonic switching has been in space and/or wavelength circuit switches (switching on timescales much larger than packet lengths) in wide area and long haul networks. However, these circuit switches demonstrate the potential for energy and latency saving having energy consumption of the order of 0.01 nJ/b [111] (approximately three orders of magnitude lower than electronic routers) when used for router bypass. In this section, we review the state of the art in photonic networking, in particular, seeking to answer two questions. First, how can we build reconfigurable photonic arrays (to work alongside reconfigurable electronic arrays such as FPGAs) in future networks? Second, how will photonic switching affect the

ability to apply the techniques of SDN in future systems? In Section VI-A and B, we describe developments in photonic integration and photonic switch technologies which will be necessary to build reconfigurable photonic systems. In Section VI-C, we contrast photonic router architectures with conventional electronic versions concentrating on the application of SDN techniques.

A. Photonic Integration

Despite the advantages of point-to-point photonic links, their use is not ubiquitous. In data centers, the links between hosts and the local rack switch are commonly copper, while photonic links replacing the copper interconnect for memories and peripherals (internal: e.g., PCI-Express, or external: e.g., USB) is rarer still. In part, this is due to the low level of integration and the consequent high cost (particularly for packaging) of photonic devices. For example, a 10-Gb/s optical transceiver may cost tens of dollars and still requires a high bandwidth, power hungry electronic link across printed circuit board tracks between the optical module, and the processor or switching device. Integrating CMOS electronics with photonics on the same chip or in the same package has been a long term industry goal in order to share the packaging cost over a larger system [112]. FPGAs already feature high-speed serial electronic transceivers, as discussed in Section V. The addition of photonic transceivers is a logical, and probably inevitable, future step. Indeed, in order to continue the increase in bandwidth of switching systems (described in Section V-A), future systems will require integrated photonics using wavelength division multiplexing (WDM). These will be used to overcome bandwidth limits due to the number of connectors fitting onto a rack unit front panel or the number of high-speed signal pins on a chip. The cost issue is also important to reconfigurability: the high cost of optical modules has meant that previous research has focused on defining photonic network architectures which minimize the number of components. However, reconfigurability inevitably requires overprovisioning (e.g., FPGA-based configurable logic blocks and transceivers) which can only occur when these components are low cost.

While serial electronic transceivers operating at up to 12.5 Gb/s have been integrated on the same die as configurable logic, the highest bit rates have been achieved by implementing the transceivers on a separate silicon die connected to the configurable logic of the FPGA via an interposer. The interposer is a carrier, usually fabricated from silicon or glass, with only passive interconnect between the active dies and the package substrate [113]. This approach enables individual functions, for example, configurable logic, memory, and serial transceivers to be fabricated using optimized processes before final integration, as well as reducing overall costs through reduced individual die area. An alternative to the interposer approach is to attach photonic transceivers directly to the CMOS chip in a 3-D arrangement [114]. Ultimately this offers minimum interconnect delays between circuit elements and maximum bandwidth between layers but is challenging due to thermal, stress, and reliability issues and requires new methods for testing.

B. Photonic Switching

Photonic switches are the key component of future reconfigurable photonic systems. High bandwidths and port count photonic switches are achieved using a combination of space switching (switching between waveguides or fibers) and wavelength switching. The differences between space and wavelength switching are illustrated in Fig. 7. Space switching allows multiple wavelengths to be switched in a single operation resulting in high bandwidth and low serialization latency. In wavelength switching, the signal wavelength defines the route through the network. However, unlike in electronic switches in which the signals are regenerated at each sequential element [register, first-in–first-out (FIFO), etc.], optical loss, crosstalk, and noise build up along a cascade of optical switches, limiting scalability of the network.

The reconfiguration time of photonic switch technologies varies over some six orders of magnitude. At millisecond to microsecond timescales, microelectromechanical systems (MEMS) [115], [116], piezoelectric [117], thermo-optic, or liquid crystal beam steering technologies are suitable for occasional or start-of-life reconfiguration. These technologies offer scalability to large port counts (up to 200 ports for piezoelectric and $>$ 1000 ports for MEMS at the current time) and low loss ($<$ 1 dB for piezoelectric devices) for transmission through a cascade of switches without electronic regeneration or amplification. These switches are already widely used in circuit switches in wide area networks and increasingly in data centers (see Section VI-C).

For compatibility with both circuit and packet switching, several devices offer multiwavelength space switching on

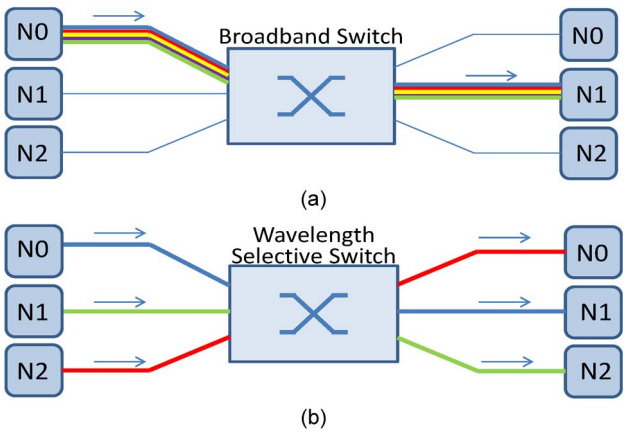

Fig. 7. *Photonic switching using (a) space switching with multiple wavelengths per port to increase port bandwidth; and (b) wavelength switching in which the signal wavelength defines its output port.*

nanosecond timescales: Mach–Zehnder interferometers (MZIs) [118], semiconductor optical amplifier (SOA) [119], and ring resonators [120]. For these switching technologies, large prototype switches have not been demonstrated, so predictions are extrapolated from smaller scale demonstrations. The gain of SOA switching elements enables losses to be overcome and scaling to thousands of ports has been demonstrated using discrete components connected by optical fiber [121]. However, in integrated circuit implementations, the higher gain required to overcome losses due to splitting and waveguide crossings causes a build up of noise, limiting scalability. Integrated SOA switches have been shown to scale to 64 ports with ten wavelengths per port [119], while a hybrid MZI and SOA integrated switch architecture has been shown to scale to 128 ports [122]. The scaling of both MZI and ring resonator switches is limited by optical losses and crosstalk. Ring resonators have attracted considerable interest due to their small physical size (down to a few micrometers diameter) and scaling to 256 ports with ten wavelengths per port has been claimed [123], but require accurate temperature control. Ring resonators can be used as both WDM space switches and wavelength routing elements. The other main option for wavelength routing is the arrayed waveguide grating (AWG) [124] which is also limited in scalability by loss and crosstalk.

As discussed in Section VI-A, reconfigurable photonic arrays will need to integrate many components on a single chip in order to reduce die costs and amortize packaging costs. Therefore, switching component dimensions are critical. Section V notes the impressive track record of smaller features and increased die sizes for standard electronics driven, at least in part, by the cost benefits of smaller circuits. While these improvements have provided partial benefits to the control circuits and DSP coupled to photonic devices, photonic systems themselves are subject to other, unique, constraints. In particular, the minimum size for photonic devices is determined by the wavelength of light: typically between 800 nm and 1.6 μm for communication systems. For example, optical waveguides must be larger than one-half of the wavelength of the light in use. This is at least an order of magnitude larger than the size of complementary metal–oxide–semiconductor (CMOS) transistors. This means there are fundamental limits on the miniaturization of photonic components and, in contrast to the scaling of CMOS devices, photonic device sizes cannot be continuously scaled in physical dimensions. A key metric determining the size of photonic circuits is the minimum achievable waveguide bend radius. In this respect, silicon is superior to traditional optical materials with minimum bend radii down to a few micrometers (due to the high contrast between the refractive index of the waveguide core and cladding materials) [125]. Silicon photonic elements have been demonstrated that are sufficiently small as to allow large integrated photonic circuits and switch fabrics. Typical dimensions for two-port silicon photonic switches are 10^4 μm^2 for MZI devices and 10^2 μm^2 for ring resonators [126] allowing, respectively, up to 10^4 or 10^6 switching elements to be implemented on a 400-mm^2 die. The area of SOAs ranges from 10^4 to 10^5 μm^2 depending on the gain required, whereas the smallest AWGs in silicon are of the order of 10^5 μm^2.

Reconfigurable photonic arrays incorporating photonic transceivers and switching elements could assist in meeting the energy and latency goals of future systems. Such arrays could provide both circuit and packet switching functionality as well as reconfigurable high-speed WDM interfaces with FPGA or other programmable electronic fabrics. As discussed in Section VI-A, high levels of photonic integration (and hence low cost per device) are in order to make this vision economically viable. The requirement rules out devices which have large physical size or high power, for example, photonic delay lines and buffers and wavelength converters based on nonlinear processes. A key question is the building block switch technology. While slow electromechanical switches could provide reconfigurability with low losses, fast electro-optic switches based on SOA, MZI, or ring resonator technologies ultimately offer greater functionality and potential for integration. However, further development is required to achieve very low losses in integrated circuit implementations to make this vision a practical reality.

C. Photonic Networks and SDN

Electronic switches and routers place packets into memory (or at least the header in cut-through switches) to permit header processing and routing decisions (described in Section IV) to be carried out. Replicating this functionality exactly in the photonic domain including packet buffering, header processing logic, and switch control, as shown in Fig. 8(a), is challenging due to the immaturity of both optical memory and logic. However, optical logic functions of Internet routers such as decrementing a packet's time to live (TTL) counter has been demonstrated [127] and continuing research aims to demonstrate routing table lookup and checksum processing in the optical domain under electronic control [128]. Unless the header is sent on a separate control channel ahead of data, optical memory or delay lines are required to delay the data while header processing is taking place. Switched fiber or integrated circuit recirculation loops have been demonstrated but these have a large physical size and are unsuitable for integration with other photonic circuits and electronics. Other studies have shown that the use of fiber delay lines is not feasible in large switching systems such as Internet routers [129]. This can be clearly understood by considering that a high-performance router used by Internet service providers will typically provide 250 ms of buffering per port; this would require 50 000 km of fiber delay lines. In general, these all-optical packet switch approaches will yield minimum latency, but will never achieve the density of electronic memory and logic due to the fundamental limitation of the wavelength of light discussed above. In

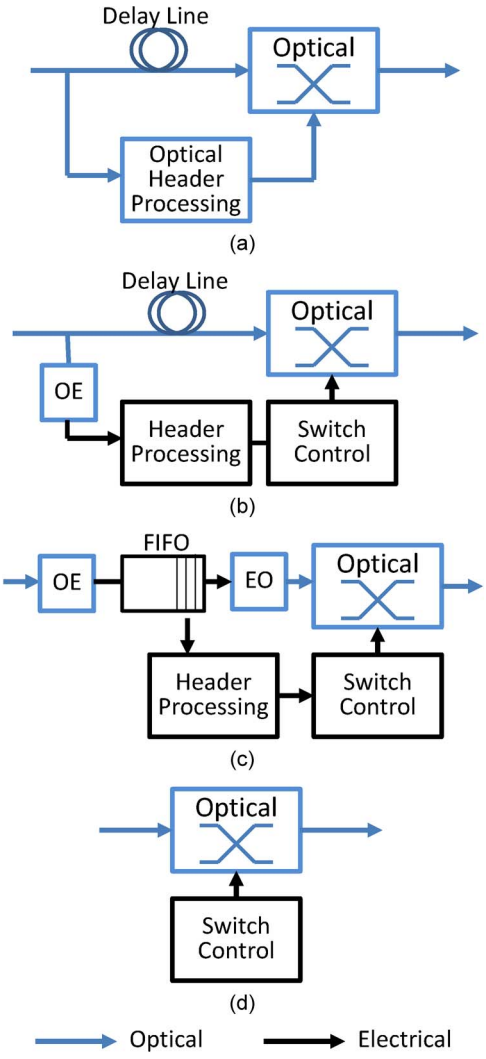

Fig. 8. ***Photonic routers. (a) All optical packet switch: may rely on electronic packet switch backup. (b) Optical packet switch: relies on electronic logic. (c) Optical–electrical–optical: relies on electronic buffering and logic. (d) Circuit switched: relies on electronic or photonic packet switching. Blue lines represent optical functions and connections. Black lines represent electronic functions and connections. OE = optical-to-electrical conversion. EO = electrical-to-optical conversion.***

addition, the optical processing functions cited above use nonlinear processes which have low energy efficiency. It is also likely that electronic routing will still be required. For example, in [128], it is proposed that only packets with a common address are routed optically (due to limitations in the number of destination fields which can be examined in an optical correlator) with other packets routed to an electronic packet switch.

Avoiding either optical memory or optical logic requires that at least the packet header is converted back to the electronic domain, as shown in Fig. 8(b). As with the all-optical approach, this requires that the header be sent ahead of the data and/or that the packet be stored in a delay line. In order to minimize this delay, fast header processing and low header serialization latency is required. Therefore, the header fields are kept to a minimum in these systems (e.g., just a valid bit plus port destination in [130] and [131]) and so the ability to differentiate actions between different packet types is extremely limited. In general, although both packet switches with all-optical or electronic header processing can achieve low latency for a limited number of cases, the ability to apply the rich header processing and SDN functions described in Section II is limited.

Several groups investigating large port count or multiple stage optical switches have come to the conclusion that conversion of the entire packet back to the electronic domain for electronic buffering and header processing is necessary [Fig. 8(c)]. This approach enables SDN approaches to be used with little modification. It is also highly scalable as the optical signals are regenerated at each optical–electrical–optical (OEO) conversion point. However, this approach reduces the potential for energy and latency benefits through photonic switching. OSMOSIS [132], an example of such an optical switch aimed at 2000-port shared memory supercomputers, introduced an OEO conversion at the input of each 64-port photonic switch stage in order to maintain short connections between electronic packet queues and the switch scheduler. Scalable Internet routers have also been proposed using OEO stages between the load balancing and switching stages [133].

Circuit switching [Fig. 8(d)] is the natural flow control mechanism for photonics and is the only area which has seen widespread commercial adoption in the form of optical cross connects (space switches) and reconfigurable optical add drop multiplexers (ROADM, space, and wavelength switches) used in core networks. In these scenarios, carriers can switch at the port or wavelength level to allow heavy traffic flows to be rerouted or to bypass electronic routers. Similar hybrid packet and circuit switching techniques have also been proposed for data centers and high-performance computing (HPC) either operated in parallel with electronic packet switches [134], [135] or as a reconfigurable physical layer for an electronic packet switch network [136]. Traditionally, the optical space and wavelength circuit switching has been managed by a separate control plane, typically based on the generalized multiprotocol label switching (GMPLS) protocol [137]. Current research investigates the use of SDN to unify the control of circuit and packet switching in a common structure. An extension to OpenFlow has been developed to handle circuit switching [138]. Other researchers propose the integration of GMPLS control structures into SDN implementations [139], [140]. There has also been a widespread move to build SDN interfaces into commercial optical circuit switching products, for example, [115]–[117], [141]. As circuit switches are designed to complement rather than replace packet switches (electronic or photonic), there is considerable scope for creating rules for packet or circuit

decisions [142] or reconfiguration to optimize for changing bandwidth demands.

VII. SUMMARY

Network systems have developed to incorporate vast depth and breadth of reconfiguration. Techniques for reconfiguration in networks span time scales, network implementation technologies, and approaches to reconfiguration itself. Into this domain, SDN has emerged as a dominant paradigm of network system reconfiguration, from the configuration of devices at setup to the reconfiguration and update of those devices over their lifetime. This paper shows the interplay between the need for reconfiguration in networks and development of SDN.

While a term only recently coined, SDN represents many decades of developments in network control through reconfiguration. Its rise in recent years has been in response to an environment that had made innovation, for both users and researchers, increasingly challenging. While SDN itself is defined a number of ways, common across all definitions is the strong isolation between different planes in the network (e.g., between the control and data plane), along with the enabling of centralized network management, and a high level of programmability.

Within this paper, we have provided a tutorial of SDN with an emphasis on OpenFlow which (at the time of this writing) was the most popular incarnation of SDN. We have argued that SDN arises to permit control and reconfiguration across devices with a forcing function being the increase in device flexibility. We have further shown that the only realistic mechanism permitting elegant interworking between the packet-centric networks of the electrical domain and the flow-centric networks of the optical domain is to permit deeper levels of network-system and device reconfiguration through SDN. Finally, we have examined the near future for both electronic and photonic reconfiguration technologies and how these will be enabled by the opportunities provided by SDN. ■

Acknowledgment

The authors would like to thank A. Wonfor, T. Moncaster, N. Sultana, G. Neville-Neil, R. Clegg, P. G. Neumann, D. Riddoch, J. Lyke, and the anonymous reviewers for helping them improve this paper. They would also like to thank C. McCabe, A. Chang, and M. Gustlin from Xilinx, G. Lange and G. Rosenfeld from Broadcom, and D. Green from EZchip for their assistance.

REFERENCES

[1] N. McKeown, "Software-defined networking," in *Proc. 28th Annu. Joint Conf. IEEE Comput. Commun. Soc.*, 2009, vol. 17, no. 2, pp. 30–32.

[2] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: An intellectual history of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–98, Apr. 2014.

[3] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[4] Open Networking Foundation (ONF), "Software-defined networking (SDN) definition." [Online]. Available: https://www.opennetworking.org/sdn-resources/sdn-definition

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

[6] I. Leslie and D. McAuley, "Fairisle: An ATM network for the local area," in *Proc. ACM SIGCOMM Commun. Architect. Protocols*, Aug. 1991, p. 327.

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

[8] I. Hadžić and J. M. Smith, "Balancing performance and flexibility with hardware support for network architectures," *ACM Trans. Comput. Syst.*, vol. 21, pp. 375–411, Nov. 2003.

[9] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an OpenFlow switch on the NetFPGA platform," in *Proc. 4th ACM/IEEE Symp. Architect. Netw. Commun. Syst.*, 2008, DOI: 10.1145/1477942.1477944.

[10] J. W. Lockwood *et al.*, "NetFPGA—An open platform for gigabit-rate network switching and routing," in *Proc. IEEE Int. Conf. Microelectron. Syst. Educ.*, 2007, pp. 160–161.

[11] M. Blott, J. Ellithorpe, N. McKeown, K. Vissers, and H. Zeng, "FPGA research design platform fuels network advances," *Xilinx Xcell J.*, no. 73, pp. 24–29, 2010.

[12] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 Gbps as research commodity," *IEEE Micro*, no. 5, pp. 32–41, Sep./Oct. 2014.

[13] F. Hu, Q. Hao, and K. Bao, "A survey on software defined networking (SDN) and OpenFlow: From concept to implementation," *IEEE Commun. Surv. Tut.*, vol. 16, no. 4, pp. 2181–2206, 4th Quart. 2014.

[14] K.-K. Yap *et al.*, "OpenRoads: Empowering research in mobile networks," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 125–126, Jan. 2010.

[15] H. Ali-Ahmad *et al.*, "CROWD: An SDN approach for densenets," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2013, pp. 25–31.

[16] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards Programmable Enterprise WLANS with Odin," in *Proc. Hot Topics Softw. Defined Netw.*, 2012, pp. 115–120.

[17] M. Stensgaard and J. Sparso, "ReNoC: A network-on-chip architecture with reconfigurable topology," in *Proc. IEEE Int. Symp. Networks-on-Chip*, Apr. 2008, pp. 55–64.

[18] Y.-C. Lan, S.-H. Lo, Y.-C. Lin, Y.-H. Hu, and S.-J. Chen, "BiNoC: A bidirectional NoC architecture with dynamic self-reconfigurable channel," in *Proc. IEEE Int. Symp. Networks-on-Chip*, May 2009, pp. 266–275.

[19] Z. Zhang, A. Greiner, and S. Taktak, "A reconfigurable routing algorithm for a fault-tolerant 2D-mesh network-on-chip," in *Proc. Design Autom. Conf.*, Jun. 2008, pp. 441–446.

[20] P. Wolkotte, G. J. M. Smit, G. Rauwerda, and L. Smit, "An energy-efficient reconfigurable circuit-switched network-on-chip," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, Apr. 2005, DOI: 10.1109/IPDPS.2005.95.

[21] M. Dillinger, K. Madani, and N. Alonistioti, *Software Defined Radio: Architectures, Systems and Functions*. New York, NY, USA: Wiley, 2003.

[22] P. Barham *et al.*, "Xen and the art of virtualization," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, 2003.

[23] European Telecommunications Standards Institute (ETSI), "Network functions virtualisation," Introductory White Paper, Jun. 2012. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[24] C. Kachris, G. C. Sirakoulis, and D. Soudris, "Network function virtualization based on FPGAs: A framework for all-programmable network devices," *Comput. Res. Repository (CoRR)*, vol. abs/1406 0309, 2014.

[25] A. Gember-Jacobson *et al.*, "OpenNF: Enabling innovation in network function control," in *Proc. ACM SIGCOMM Conf.*, Chicago, IL, USA, Aug. 2014, pp. 163–174.

[26] Open Networking Foundation (ONF), "SDN architecture," Jul. 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf

[27] G. Varghese, *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*. San Mateo, CA, USA: Morgan Kaufmann, 2004.

[28] Infinispan. [Online]. Available: http://infinispan.org/

[29] OpenDaylight, "A Linux Foundation collaborative project." [Online]. Available: http://www.opendaylight.org

[30] M. Casado *et al.*, "Ethane: Taking control of the enterprise," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 1–12, Aug. 2007.

[31] N. Gude *et al.*, "NOX: Towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[32] ONOS, "A new carrier-grade SDN network operating system designed for high availability, performance, scale-out." [Online]. Available: http://onosproject.org/

[33] Ryu, "SDN framework." [Online]. Available: http://osrg.github.io/ryu/

[34] Floodlight, "OpenFlow controller—Project Floodlight." [Online]. Available: http://www.projectfloodlight.org/floodlight/

[35] VMware, "Network virtualization with VMware NSX virtualized network." [Online]. Available: https://supportforums.cisco.com/document/105496/asr9000xr-understanding-route-scale

[36] C. Monsanto, N. Foster, R. Harrison, and D. Walker, "A compiler and run-time system for network programming languages," *ACM SIGPLAN Notices*, vol. 47, no. 1, pp. 217–230, 2012.

[37] A. Voellmy and P. Hudak, "Nettle: Taking the sting out of programming network routers *Practical Aspects of Declarative Languages*. New York, NY, USA: Springer-Verlag, 2011, pp. 235–249.

[38] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: Simplifying SDN programming using algorithmic policies," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 87–98.

[39] N. P. Katta, J. Rexford, and D. Walker, "Logic programming for software-defined networks," in *Proc. Workshop Cross-Model Lang. Design Implement.*, pp. 1–3, 2012.

[40] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Practical declarative network management," in *Proc. ACM SIGCOMM Workshop, Res. Enterprise Netw.*, 2009, DOI: 10.1145/1592681.1592683.

[41] N. Foster *et al.*, "Frenetic: A network programming language," in *Proc. 16th ACM SIGPLAN Int. Conf. Functional Programm.*, 2011, pp. 279–291.

[42] C. Monsanto *et al.*, "Composing software defined networks," in *Proc. 10th USENIX Conf. Netw. Syst. Design Implement.*, 2013, pp. 1–13.

[43] R. Soulé, S. Basu, R. Kleinberg, E. G. Sirer, and N. Foster, "Managing the network with Merlin," in *Proc. 12th ACM Workshop Hot Topics Netw.*, 2013, DOI: 10.1145/2535771.2535792.

[44] A. Voellmy, H. Kim, and N. Feamster, "Procera: A language for high-level reactive network control," in *Proc. Hot Topics Softw. Defined Netw.*, 2012, pp. 43–48.

[45] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An open framework for OpenFlow switch evaluation," in *Proc. 13th Int. Conf. Passive Active Meas.*, 2012, pp. 85–95.

[46] P. Francois, M. Shand, and O. Bonaventure, "Disruption free topology reconfiguration in OSPF networks," in *Proc. 26th Annu. Joint Conf. IEEE Comput. Commun. Soc.*, May 2007, pp. 89–97.

[47] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, "Consistent updates for software-defined networks: Change you can believe in!" in *Proc. 10th ACM Workshop Hot Topics Netw.*, 2011, DOI: 10.1145/2070562.2070569.

[48] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Architect. Protocols Comput. Commun.*, 2012, pp. 323–334.

[49] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *Proc. Hot Topics Softw. Defined Netw.*, 2013, pp. 49–54.

[50] Broadcom, "OpenFlow data-plane abstraction (OF-DPA): Abstract switch specification," 2014. [Online]. Available: http://www.broadcom.com/collateral/etp/OFDPA_OASS-ETP101-R.pdf

[51] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *Proc. Hot Topics Softw. Defined Netw.*, 2013, pp. 127–132.

[52] P. Bosshart *et al.*, "Programming protocol-independent packet processors," *Comput. Res. Repository (CoRR)*, vol. abs/1312 1719, 2013.

[53] The P4 Language Consortium. [Online]. Available: http://www.p4.org

[54] Marvell, "Marvel Xelerated HX4100 family of network processors," Product Brief. [Online]. Available: http://www.marvell.com/network-processors/assets/Marvell_Xelerated_HX4100-02_product

[55] R. Giladi, *Network Processors: Architecture, Programming, Implementation*. San Mateo, CA, USA: Morgan Kaufmann, 2008.

[56] Cisco, "The Cisco QuantumFlow Processor: Cisco's next generation network processor." [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/routers/asr-1000-series-aggregation-services-routers/solution_overview_c22-448936.pdf

[57] B. Wheeler, "A new era of network processing," The Linley Group, Tech. Rep., 2013.

[58] EZChip, "Building scalable and efficient SDN and NFV architectures." [Online]. Available: http://www.ezchip.com/Images/pdf/EZchip_SDN_&_NFV_Oct2013.pdf

[59] R. Ennals, R. Sharp, and A. Mycroft, "Task partitioning for multi-core network processors," *Compiler Construct.*, vol. 3443, pp. 76–90, 2005.

[60] G. Brebner and W. Jiang, "High-speed packet processing using reconfigurable computing," *IEEE Micro*, vol. 34, no. 1, pp. 8–18, Jan./Feb. 2014.

[61] M. Attig and G. J. Brebner, "400 Gb/s programmable packet parsing on a single FPGA," in *Proc. 7th ACM/IEEE Symp. Architect. Netw. Commun. Syst.*, 2011, pp. 12–23.

[62] J. Bachrach *et al.*, "Chisel: Constructing hardware in a scala embedded language," in *Proc. Design Autom. Conf.*, 2012, pp. 1216–1225.

[63] MicroJamJar, "HardCaml: Register transfer level hardware design in OCaml." [Online]. Available: http://ujamjar.github.io/hardcaml

[64] Open Network Foundation (ONF). [Online]. Available: http://www.opennetworking.org

[65] Internet Engineering Task Force (IETF), "IRTF—Software defined networking research group (SDNRG)." [Online]. Available: https://trac.tools.ietf.org/group/irtf/trac/wiki/sdnrg

[66] Cisco, "Open network environment for service providers." [Online]. Available: http://www.cisco.com/c/en/us/solutions/service-provider/open-network-environment-service-providers/index.html

[67] Cisco, "Cisco unveils nPower, world's most advanced network processor." [Online]. Available: http://newsroom.cisco.com/release/1262342

[68] Cisco, "ASR9000/XR understanding route scale." [Online]. Available: https://supportforums.cisco.com/document/105496/asr9000xr-understanding-route-scale

[69] Cisco, "The Cisco QuantumFlow Processor: Cisco's next generation network processor." [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/routers/asr-1000-series-ag gregation-services-routers/solution_overview_c22-448936.html

[70] Cisco, "Cisco's One Platform Kit (onePK)." [Online]. Available: http://www.cisco.com/c/en/us/products/ios-nx-os-software/onepk.html

[71] P. J. Leach *et al.*, *RFC 2616: Hypertext Transfer Protocol—HTTP/1.1*, 1999.

[72] J. Ungerman, "OpenFlow," Cisco Connect, 2014.

[73] Xilinx, "Software defined specification environment for networking." [Online]. Available: http://www.xilinx.com/applications/wired-communications/sdnet.html

[74] Marvell, "Data flow architecture." [Online]. Available: http://www.marvell.com/network-processors/technology/data-flow-architecture/

[75] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Trans. Netw.*, vol. 1, no. 3, pp. 344–357, Jun. 1993.

[76] A. K. Parekh and R. G. Gallagher, "A generalized processor sharing approach to flow control in integrated services networks: The multiple node case," *IEEE/ACM Trans. Netw.*, vol. 2, no. 2, pp. 137–150, Apr. 1994.

[77] K. Bala, I. Cidon, and K. Sohraby, "Congestion control for high speed packet switched networks," in *Proc. 9th Annu. Joint Conf. IEEE Comput. Commun. Soc.*, 1990, pp. 520–526.

[78] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[79] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ECN) to IP," RFC 3168, Sep. 2001, Updated by RFCs 4301, 6040.

[80] *IEEE Standard for Local and metropolitan area networks- Virtual Bridged Local Area Networks Amendment 13: Congestion Notification*, IEEE Std. 802.1Qau-2010 (Amendment to IEEE Std. 802.1Q-2005), Apr. 2010, pp. c1-119.

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

[82] D. Palma *et al.*, "The queuepusher: Enabling queue management in OpenFlow," in *Proc. Eur. Workshop Softw. Defined Netw.*, 2014, pp. 125–126.

[83] A. Sivaraman, K. Winstein, S. Subramanian, and H. Balakrishnan, "No silver bullet: Extending SDN to the data-plane," in *Proc. 12th ACM Workshop Hot Topics Netw.*, 2013, DOI: 10.1145/2535771.2535796.

[84] H. Farhad, H. Lee, and A. Nakao, "Data plane programmability in SDN," in

Proc. IEEE 22nd Int. Conf. Netw. Protocols, Oct. 2014, pp. 583–588.

[85] Cisco, "Cisco's massively scalable data center, network fabric for warehouse scale computer," Tech. Rep., 2010, pp. 583–588.

[86] V. Mehta, "New IBM system networking innovations address CIOs' pain points," IBM, Tech. Rep., 2011.

[87] Broadcom, "Broadcom enables massive network scalability with world's highest density 100GbE switch solution," Apr. 2012. [Online]. Available: http://investor.broadcom.com/releasedetail.cfm?ReleaseID=668485

[88] Cisco, "Building cost effective and scalable CORE networks using an elastic architecture," Tech. Rep., 2013.

[89] Huawei, "NE5000E." [Online]. Available: http://huawei.com/us/products/data-communication/ne-routers/ne5000e/index.htm

[90] Cisco, "Cisco adds carrier routing system X (CRS-X) core router to industry-leading CRS family." [Online]. Available: http://newsroom.cisco.com/release/1208192

[91] Altera, "Embedded processors." [Online]. Available: http://www.altera.co.uk/products/ip/processors/ipm-index.jsp

[92] Xilinx, "Embedded processing peripheral IP cores." [Online]. Available: http://www.xilinx.com/ise/embedded/edk_ip.htm

[93] B. H. Fletcher, "FPGA embedded processors, revealing true system performance," in *Proc. Embedded Syst. Conf.*, pp. 1–18, 2005.

[94] Altera, "Dual-core ARM Cortex-A9 MPCore processor." [Online]. Available: http://www.altera.co.uk/devices/processor/arm/cortex-a9/m-arm-cortex-a9.html

[95] Xilinx, "MicroBlaze soft processor core." [Online]. Available: http://www.xilinx.com/tools/microblaze.htm

[96] Altera, "Nios II processor reference handbook," Tech. Rep. NII5V1-13.1, 2014.

[97] H. Subramoni, F. Petrini, V. Agarwal, and D. Pasetto, "Intra-socket and inter-socket communication in multi-core systems," *Comput. Architect. Lett.*, vol. 9, pp. 13–16, Jan. 2010.

[98] Intel, "Intel Omni-Path architecture." [Online]. Available: http://www.intel.com/content/www/us/en/omni-path/omni-path-fabric-overview. html

[99] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Comput. Architect. News*, vol. 34, no. 4, pp. 1–17, 2006.

[100] Altera, "Stratix 10 FPGAs and SoCs: Delivering the unimaginable." [Online]. Available: http://www.altera.co.uk/devices/fpga/stratix-fpgas/stratix10/stx10-index.jsp

[101] M. Gustlin, F. Olsson, and M. Weber, "Interlaken technology: New-generation packet interconnect protocol," White Paper, Mar. 2007. [Online]. Available: https://www.cortina-systems.com/images/documents/400023_Interlaken_Technology_White_Paper.pdf

[102] R. Cam, R. T. Lerer, K. Gass, and W. Nation, "System packet interface level 4 (SPI-4) phase 2 revision 1: OC-192 system interface for physical and link layer devices," Implementation Agreement: OIF-SPI4-02.1, 2003.

[103] Optical Internetworking Forum (OIF), "Next generation interconnect framework," Tech. Rep. OIF-FD-Client-400G/1T-01.0, 2013.

[104] A. Torza, "Using FPGA technology to solve the challenges of implementing high-end networking equipment: Adding a 100 GbE MAC to existing telecom equipment," Xilinx, Tech. Rep. WP280, 2008.

[105] Xilinx, "Xilinx highlights all programmable solutions for 400GE applications at WDM Nice," 2014. [Online]. Available: http://press.xilinx.com/2014-06-17-Xilinx-Highlights-All-Programmable-Solut ions-for-400GE-Applications-at-WDM-Nice-2014

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

[107] Intel, "Intel Xeon processor E7 v2 product family." [Online]. Available: http://newsroom.intel.com/docs/DOC-5073

[108] Xilinx, "Xilinx ships world's highest capacity FPGA and shatters industry record for number of transistors by 2X," 2011. [Online]. Available: http://press.xilinx.com/2011-10-25-Xilinx-Ships-Worlds-Highest-Capacity-FPG A-and-Shatters-Industry-Record-for-Number-of-Transistors-by-2X

[109] Broadcom, "Broadcom delivers industry's first high-density 25/100 gigabit Ethernet switch for gloud-scale networks," Sep. 2014. [Online]. Available: http://www.broadcom.com/press/release.php?id=s872349

[110] D. A. B. Miller, "Device requirements for optical interconnects to silicon chips," *Proc. IEEE*, vol. 97, no. 7, pp. 1166–1185, Jul. 2009.

[111] R. S. Tucker, "Green optical communications—Part I: Energy limitations in transport," *IEEE J. Sel. Top. Quantum Electron.*, vol. 17, no. 2, pp. 245–260, Mar./Apr. 2010.

[112] U. Vlasov, "Silicon photonics for next generation computing systems," in *Proc. Eur. Conf. Opt. Commun.*, 2008, DOI: 10.1109/ECOC.2008.4729553.

[113] K. Saban, "Xilinx stacked silicon interconnect technology delivers breakthrough FPGA capacity, bandwidth, power efficiency," Xilinx, Tech. Rep. WP380, 2012.

[114] P. Duan, O. Raz, B. Smalbrugge, and H. J. S. Dorren, "Demonstration of wafer scale fabrication of 3D stacked transmitter and receiver modules for optical interconnects," *J. Lightw. Technol.*, vol. 31, no. 24, pp. 4073–4079, Dec. 15, 2013.

[115] G. C. Solutions, "Enterprise data center." [Online]. Available: http://www.glimmerglass.com/solutions/enterprise-data-center/

[116] Calient, "The software defined hybrid packet optical datacenter network," 2013. [Online]. Available: http://www.calient.net/solutions/software-defined-datacenter-networks/

[117] Polatis, "All-optical switching and software defined networking (SDN) in today's evolving data center," SDN Appl. Note.

[118] B. Lee *et al.*, "Monolithic silicon integration of scaled photonic switch fabrics, CMOS logic, device driver circuits," *J. Lightw. Technol.*, vol. 32, no. 4, pp. 743–751, Feb. 15, 2014.

[119] A. Wonfor, H. Wang, R. V. Penty, and I. H. White, "Large port count high-speed optical switch fabric for use within datacenters," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 3, no. 8, pp. A32–A39, Aug. 2011.

[120] A. W. Poon, X. S. Luo, F. Xu, and H. Chen, "Cascaded microresonator-based matrix switch for silicon on-chip optical interconnection," *Proc. IEEE*, vol. 97, no. 7, pp. 1216–1238, Jul. 2009.

[121] O. Liboiron-Ladouceur, B. A. Small, and K. Bergman, "Physical layer scalability of WDM optical packet interconnection networks," *J. Lightw. Technol.*, vol. 24, no. 1, pp. 262–270, Jan. 2006.

[122] Q. Cheng, A. Wonfor, J. L. Wei, R. V. Penty, and I. H. White, "Demonstration of the feasibility of large port count optical switching using a hybrid MZI-SOA switch module in a recirculating loop," *Opt. Lett.*, vol. 39, no. 18, pp. 5244–5247, Sep. 2014, DOI: 10.1364/OL.39.005244.

[123] A. Biberman *et al.*, "CMOS-compatible scalable photonic switch architecture using 3D-integrated deposited silicon materials for high-performance data center networks," in *Proc. Opt. Fiber Commun. Conf.*, Mar. 2011.

[124] L. Wang *et al.*, "Athermal arrayed waveguide gratings in silicon-on-insulator by overlaying a polymer cladding on narrowed arrayed waveguides," *Appl. Opt.*, vol. 51, pp. 1251–1256, Mar. 2012.

[125] M. Lipson, "Guiding, modulating, emitting light on silicon—Challenges and opportunities," *J. Lightw. Technol.*, vol. 23, no. 12, pp. 4222–4238, Dec. 2005.

[126] G. T. Reed, G. Mashanovich, F. Y. Gardes, and D. J. Thomson, "Silicon optical modulators," *Nature Photon.*, vol. 4, no. 8, pp. 518–526, 2010.

[127] J. McGeehan *et al.*, "All-optical decrementing of a packet's time-to-live (TTL) field and subsequent dropping of a zero-TTL packet," *J. Lightw. Technol.*, vol. 21, no. 11, pp. 2746–2752, Nov. 2003.

[128] J. Touch, J. Bannister, S. Suryaputra, and A. Willner, "A design for an Internet router with a digital optical data-plane," *Proc. SPIE—Int. Soc. Opt. Eng.*, vol. 9008, pp. 9–15, 2013, DOI: 10.1117/12.2041127.

[129] R. S. Tucker, "The role of optics and electronics in high-capacity routers," *J. Lightw. Technol.*, vol. 24, no. 12, pp. 4655–4673, Dec. 2006.

[130] O. Liboiron-Ladouceur *et al.*, "The data vortex optical packet switched interconnection network," *J. Lightw. Technol.*, vol. 26, no. 13, pp. 1777–1789, Jul. 1, 2008.

[131] A. Shacham and K. Bergman, "Building ultralow-latency interconnection networks using photonic integration," *IEEE Micro*, vol. 27, no. 4, pp. 6–20, Jul./Aug. 2007.

[132] R. Luijten, C. Minkenberg, R. Hemenway, M. Sauer, and R. Grzybowski, "Viable opto-electronic HPC interconnect fabrics," in *Proc. ACM/IEEE Conf. Supercomput.*, 2005, DOI: 10.1109/SC.2005.78.

[133] I. Keslassy *et al.*, "Scaling internet routers using optics," in *Proc. ACM SIGCOMM Conf.*, 2003, pp. 189–200.

[134] N. Farrington *et al.*, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *Proc. ACM SIGCOMM Conf.*, 2010, pp. 339–350.

[135] G. Porter *et al.*, "Integrating microsecond circuit switching into the data center," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 447–458.

[136] S. Kamil, "Reconfigurable hybrid interconnection for static and dynamic scientific applications," in *Proc. 4th Int. Conf. Comput. Front.*, 2007, pp. 183–194.

[137] E. Mannie, "Generalized multi-protocol label switching (GMPLS) architecture," RFC 3945, 2004.

[138] V. Gudla *et al.*, "Experimental demonstration of OpenFlow control of packet and circuit switches," in *Proc. Opt. Fiber Commun. Conf.*, 2010, pp. 1–3.

[139] M. Shirazipour, W. John, J. Kempf, H. Green, and M. Tatipamula, "Realizing packet-optical integration with SDN and OpenFlow 1.1 extensions," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2012, pp. 6633–6637.

[140] S. Azodolmolky *et al.*, "Integrated OpenFlow-GMPLS control-plane: An overlay model for software defined packet over optical networks," *Opt. Exp.*, vol. 19, no. 26, pp. B421–B428, 2011.

[141] A. Autenrieth and J.-P. Elbers, "Network virtualization and SDN/OpenFlow for optical networks—EU project OFELIA, ADVA optical networking SE," in *Proc. Netw. Syst.*, Mar. 2013, [Online]. http://www.fp7-ofelia.eu/assets/Publications-and-Presentations/NetSys2013-Autenrieth-Slides.pdf

[142] J. Perelló *et al.*, "All-optical packet/circuit switching-based data center network for enhanced scalability, latency, throughput," *IEEE Network*, vol. 27, no. 6, pp. 14–22, Nov. 2013.

ABOUT THE AUTHORS

Noa Zilberman (Senior Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in electrical engineering from Tel-Aviv University, Tel-Aviv, Israel, in 2003, 2007, and 2014, respectively.

She is a Research Associate in the Systems Research Group, Computer Laboratory, University of Cambridge, Cambridge, U.K. Since 1999, she has filled several development, architecture, and managerial roles in the telecommunications and semiconductor industries. In her last role, she was a Senior Principal Chip Architect at Broadcom. Her research interests include open-source research using the NetFPGA platform, network and computer architectures, high-speed interfaces, Internet measurements, and topology.

Charalampos Rotsos (Member, IEEE) received the B.Sc. degree in computer science from the University of Piraeus, Piraeus, Greece, in 2006, the M.Sc. degree in data communications, networks and distributed systems from the University College London (UCL), London, U.K., in 2007, and the Ph.D. degree in computer science from the Computer Laboratory, University of Cambridge, Cambridge, U.K., in 2015.

He is currently a Senior Research Associate at the School of Computing and Communications, Lancaster University, Lancaster, U.K., and he is part of the Network Research Group. His current research interests include software-defined networking, network experimentation, network measurement, and traffic classification.

Philip M. Watts (Member, IEEE) received the B.Sc. degree in applied physics from the University of Nottingham, Nottingham, U.K., in 1991, the M.Sc. degree in technologies for broadband communications from the University College London (UCL), London, U.K., in 2003, and the Ph.D. degree in electronic engineering from UCL in 2008.

He was with the BAE Systems Advanced Technology Centre, Chelmsford, U.K., from 1991 to 2000. From 2000 to 2010, he was a Senior Optical Hardware Engineer with Nortel Networks, Harlow, U.K.; a Researcher with Intel Research; and a Consultant with Azea Networks and Huawei Technologies. From 2008 to 2010, he was a Research Fellow with the Computer Laboratory, University of Cambridge, Cambridge, U.K. He is currently an EPSRC Research Fellow and a Lecturer with the Department of Electronic and Electrical Engineering, UCL. His current research interests include optical networks for future computer systems and control and signal processing circuits for optical communications.

Andrew W. Moore received the B.S. and M.S. degrees in digital technology and computing from Monash University, Melbourne, Australia, in 1992 and 1994, respectively, and the Ph.D. degree in computer science from the Computer Laboratory, Cambridge University, Cambridge, U.K., in 2001.

He is a Senior Lecturer at the Computer Laboratory, University of Cambridge, Cambridge, U.K., where he jointly leads the Systems Research Group working on issues of network and computer architecture. His research interests include enabling open-network research and education using the NetFPGA platform; other research pursuits include software–hardware codesign, low-power energy-aware networking, and novel network and systems data-center architectures. He leads the NetFPGA project providing an open-source reconfigurable hardware/software platform for networking research and teaching. He has been principal investigator on research grants from the Engineering and Physical Sciences Research Council (EPSRC, part of the U.K. research council), the European Union (EU), the Defense Advanced Research Projects Agency (DARPA), and the National Science Foundation (NSF) as well as collaborations with industry partners Xilinx, Cisco, Netronome, and Solarflare.