

Review Article

State of the Art and Recent Research Advances in Software Defined Networking

Taimur Bakhshi

Center for Security, Communications and Network Research, University of Plymouth, Plymouth PL4 8AA, UK

Correspondence should be addressed to Taimur Bakhshi; taimur.bakhshi@plymouth.ac.uk

Received 29 July 2016; Revised 11 October 2016; Accepted 20 October 2016; Published 15 January 2017

Academic Editor: Giovanni Pau

Copyright © 2017 Taimur Bakhshi. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Emerging network services and subsequent growth in the networking infrastructure have gained tremendous momentum in recent years. Application performance requiring rapid real-time network provisioning, optimized traffic management, and virtualization of shared resources has induced the conceptualization and adoption of new networking models. Software defined networking (SDN), one of the predominant and relatively new networking paradigms, seeks to simplify network management by decoupling network control logic from the underlying hardware and introduces real-time network programmability enabling innovation. The present work reviews the state of the art in software defined networking providing a historical perspective on complementary technologies in network programmability and the inherent shortcomings which paved the way for SDN. The SDN architecture is discussed along with popular protocols, platforms, and existing simulation and debugging solutions. Furthermore, a detailed analysis is presented around recent SDN development and deployment avenues ranging from mobile communications and data centers to campus networks and residential environments. The review concludes by highlighting implementation challenges and subsequent research directions being pursued in academia and industry to address issues related to application performance, control plane scalability and design, security, and interdomain connectivity in the context of SDN.

1. Introduction

Software defined networking (SDN) is a relatively new paradigm introduced in the world of computer networking promising a fundamental shift in the way network configuration and real-time traffic management is performed. While the term itself is quite new, the salient history of SDN can be traced back to the roots of several traffic engineering and network control mechanisms developed through the years [1–4]. The underlying objective of deriving centralizing network control primitives has always been to improve the overall network performance and to introduce some degree of network control in at least a particular segment of a much larger network. SDN is seen by many in industry and academia as culmination of these efforts. The Open Networking Foundation (ONF) [5], industry consortia furthering the work in several of areas of SDN development, defines the term SDN as “the physical separation of the network control plane from the forwarding plane and where a control plane controls several devices” [1]. The SDN framework

tends to make the data plane completely programmable and separated from the control logic and, therefore, eliminates the existing manually intensive regime of fine tuning individual hardware components. The paradigm introduces a centralized control structure which dynamically configures and governs all underlying hardware based on end user application requirements. Software developers can utilize the high level of network abstraction offered via the control plane to define sophisticated network resource utilization models and optimize the underlying network fabric according to evolving service requirements. The resulting ease in management of diverse set of network appliances according to real-time traffic conditions provides substantial benefits to operators and managers in efficiently provisioning resources as well as introducing technological and business updates in a seamless fashion. SDN is hence attracting substantial attention from both academics and industry professionals. In addition to ONF industry conglomerate, the OpenFlow Network Research Center (ONRC) was created to particularly focus academic research in SDN [6], with major standard providers

such as ETSI, IETF, ONF, 3GPP, and IEEE itself, working towards standardizing different SDN technicalities. However, despite the stated advantages and the promise of simplified management, the SDN framework encounters challenges in practical implementation hampering its functionality and resulting performance in avenues ranging from the cloud to data center networking.

The present paper highlights the state of the art in software defined networking by providing a brief historical perspective of the field as well as detailing the SDN architecture. Furthermore, compared to similar work in [3, 7–10] describing state of the art in SDN, recent deployments and operational challenges are discussed in detail to give readers a comprehensive understanding of evolving implementation avenues and subsequent studies examining scalability and real-time latency, robustness, design updates, and security challenges in software defined networking. The rest of this paper is organized as follows. Section 2 provides a brief background to SDN, legacy programmable networks, and supporting technologies as well as an appraisal of emerging service requirements cumulatively addressed by the SDN framework. Section 3 discusses the SDN architecture along with an appreciation of prominent protocols and controllers presently being deployed. Section 4 details SDN simulation, development, and debugging tools. Section 5 summarizes the progress in several SDN application avenues such as data centers, campus environments, residential networks, and wireless communications. Section 6 gives a perspective on research challenges and recent advances made in SDN application performance, scalability, design, security, and interdomain routing. Final conclusions are drawn in Section 7.

2. Background and Complementary Technologies

It is rather difficult to examine the etymology of “software defined networking” as the fundamental requirement of introducing network programmability has been around since the inception of computer networks. The term, however, was first coined in an article in 2009 [11] to describe work done in developing a standard called OpenFlow giving network engineers access to flow tables in switches and routers from external computers for changing network layout and traffic flow in real time. Technologies supporting the centralization of network control, introducing programmability and virtualization, have, however, existed prior to SDN and over the years matured to varying degrees of adoption among operators catering for individual application requirements. A timeline depicting development of key technologies highlighting efforts in centralizing network control, network programmability, and resource virtualization is presented in Figure 1. A summarization categorizing the respective technologies is provided in Table 1. The following subsections briefly discuss some of these key supporting technologies to provide a better understanding of their similarities and inadequacies in comparison with SDN.

2.1. Centralized Network Control. Centralization of network control could at least be dated back to the early 1980s where AT&T introduced the network control point (NCP) [12] offering a centralized database of telephone circuits and out-of-band signalling mechanism for calling card machines. The idea of control and data plane separation was also used in BSD4.4 routing sockets in the early 1990s, allowing route tables to be controlled by a simple command line or by a router daemon [13]. Another significant milestone in the development of centralized network control includes the Forwarding and Control Element Separation (ForCES) project, which started as an IETF working group in 2001. ForCES employs a control element to formulate the routing table in traffic forwarding elements [14]. Each control element interacts with one or more forwarding elements, in turn managed by a control and forwarding element manager offering increased scalability. With the development and wider adoption of generalized and multiprotocol label switching (G/MPLS), the routers required complex computations involving path determination satisfying multiple constraints ranging from backup route calculations to using paths conforming to a given bandwidth [15]. Individual routers, however, lacked the computing power or network knowledge to carry out the required route construction. The IETF path computation engine (PCE) working group as a consequence developed a set of protocols that allow path computation by a client such as a router to get path information from the computation engine which may be centralized or partly distributed in every router [16]. The technology has attracted significant attraction having more than twenty-five RFCs at the time of writing. The scheme, however, lacks a dedicated control or path computation engine discovery mechanism and provides only reactive or on-demand facilitation of information to computation clients. The Open Signalling (OPEN-SIG) working group started in 1995 aiming to make the ATM, Internet, and mobile networking both programmable and open [17]. The group worked towards allowing access to network hardware via programmable interfaces offering distributed and scalable deployment of new services on existing devices. The IETF took this idea to standardize and specify the General Switch Management Protocol (GSMP), a protocol managing network switch ports, connections, and monitoring statistics as well as updating and assigning switch resources via a controller [18]. The 4D project, initiated in 2004, proposed network design that separated the traffic forwarding logic and the protocols used for interelement communication [19]. The framework proposed the control or “decision” plane having a global view and catered by planes further down the hierarchy responsible for providing element state information and forwarding traffic. More recently, and a direct predecessor to enabling SDN technology was the Ethane project [20]. Proposed in 2007, the domain controller in Ethane computed flow table entries based on access control policies and used custom switches running on OpenWRT [21], NetFPGA [22], and Linux systems to implement the traffic forwarding constructs. Due to the constraints of having customized hardware, Ethane, however, was not taken up by many industry vendors as anticipated. In comparison, the present scheme for SDN uses existing hardware and

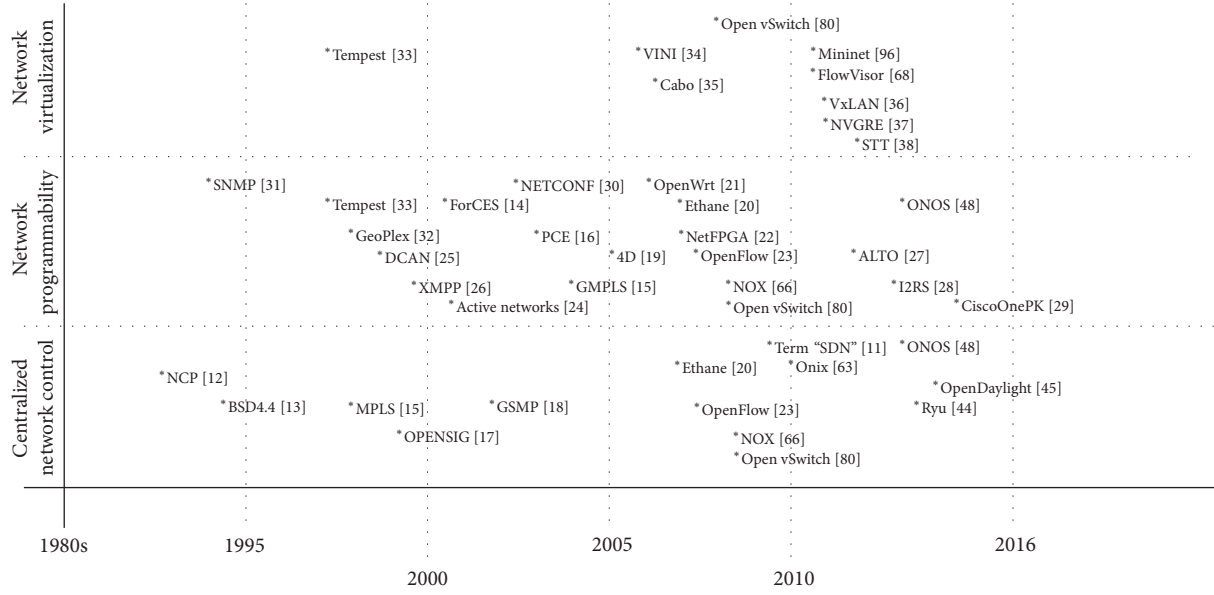


FIGURE 1: Diagram illustrating key developments in complementary networking technologies.

TABLE 1: Complementary technologies.

Functionality	Control functions, APIs	Complementary technologies
Centralized control	Centralized/delegated control framework	NCP [12], BSD4.4 Routing Sockets [13], ForCES [14], PCE [16], OPENSIG [17], GSMP [18], 4D [19], Ethane [20], G/MPLS [15]
Network programmability	Low-level network abstraction	Active Networks [24], XMPP [26], DCAN [25]
	High-level network abstraction	ALTO [27], I2RS [28], Cisco onePK [29]
	Configuration API	NETCONF [30], SNMP [31], GeoPlex [32]
Virtualization	Network device virtualization and overlays	Tempest [33], VINI [34], Cabo [35], VxLAN [36], NVGRE [37], STT [38]

vendors are only required to expose interfaces to flow tables on switches with OpenFlow [23] providing controller-switch communication capability. Growth in centralized network control has not been in isolation with parallel efforts in bringing automation and programmability to the network appliances examined in the following section.

2.2. Network Programmability. Network administrators have long yearned for increased programmability of network devices, as the present method of configuration (mainly via CLI) despite being effective is rather slow and requires laborious work in changing configurations growing significantly with the size of the network. The US Defense and Advanced Research Projects Agency (DARPA) envisioned the underlying problems in integrating new technology in conventional networking and the elaborate and tedious reconfigurations required hampering acceleration of innovation in the mid-1990s. The term active networks was proposed around the same time and advocated custom computations on packets to significantly reduce programmability of individual devices [24]. An example of this would have been trace programs running on routers and the idea of active nodes downloading new service instructions to serve as firewall or offer other

application services. However, not having a clear application at the time such as cloud services today and lack of cheap network support, the idea did not fully achieve fruition. Another network programming initiative in the mid-1990s was the Devolved Control of ATM Networks (DCAN) [25]. The underlying aim of DCAN was the designing and development of infrastructure and services required to achieve scalability in controlling and programming ATM networks. The working principle of DCN technology is that ATM switch control decisions should be decoupled from the devices and delegated to external entities, the DCAN manager. The DCAN manager in turn uses programming instructions to manage the network elements, quite similar to present day SDN. Another similar project aimed at incorporating programmability in the network elements was AT&T's GeoPlex [20]. The project utilized Java programming language to implement middleware functionality in networking. GeoPlex was meant to be a service platform managing networks and services using the operating systems running on Internet connected computers. The resulting soft switch abstraction, however, could not reprogram physical devices due to compatibility with proprietary operating systems running on these devices. Another vital addition to network programmability came in

the form of the extensible message and presence protocol (XMPP) described in RFC 6121 which works quite similar to SMTP but targeted at near real-time communication offering additional functionalities of monitoring presence along with messaging [26]. Each XMPP client sets up a connection with the server in the network which maintains contact addresses of clients and may let others know when a contact is online. Messages are pushed (real-time) as opposed to polled in SMTP/POP and the protocol is now being used in data center networking as well as the upcoming Internet of Things (IoT) paradigm to manage network elements. Network clients have XMPP clients which respond to XMPP messages containing CLI management requests. Given its ubiquity of use in legacy computing systems, XMPP has found considerable traction in the SDN (southbound API) domain.

Offering an even higher level of abstraction from a network administrator or service provider's perspective, the Application Layer Traffic Optimization (ALTO) protocol started by an IETF working group and originally aimed at optimizing P2P traffic by identifying nearby peers has seen further extension for locating resources in data centers [27]. ALTO clients produce a list of resources, inherent constraints such as memory, storage, and bandwidth and power consumption, and present this information to the server which gathers knowledge about the available resources and produces a detailed orchestration for the running applications. Another routing strategy proposing splitting of decision making between centralized management and applications is the Interface to the Routing System (I2RS) project of IETF [28]. Unlike a strict centralized SDN, I2RS proposes using traditional routing protocols executed on network hardware in parallel with centralized control. The scheme allows using distributed routing while allowing individual applications to influence routing decisions as required. Developments in network programmability, however, have not been limited solely to standardization bodies and workgroups. Technology vendors such as Cisco have also sought to enable programmers to develop application that can integrate with the network fabric. The Cisco Open Network Environment Platform Kit (Cisco onePK) provides a programmable framework allowing operators to customize traffic flows and visualize network information for easier deployment according to changing business needs [29]. The framework is now being folded into Cisco's Application Centric Infrastructure (ACI) [39] which seeks to further integrate software and hardware driven by operational requirements.

2.3. Network Virtualization. Network virtualization can be described as the representation of one or more network topologies residing on the same infrastructure. Virtualization has seen various phases of installation from being basic VLANs, to various intermediate technologies and test beds. A few milestone projects worth mentioning include Tempest, VINI, and Cabo. Tempest originated at Cambridge in 1998 and proposed the idea of a separation of control framework from switches as well as switch virtualization [33]. Tempest proved to be an early attempt at decoupling traffic forwarding and control logic, specifically in the context of ATM networks. Similar to present day SDN, Tempest

project put emphasis on having open programming interfaces and additional support for network virtualization. On a slightly separate strand focusing on testing new protocols and services, the virtual network infrastructure (VINI) came to light in 2006 offering researchers a virtual networking test bed to deploy and evaluate multiple ideas simultaneously on different network topologies using realistic routing software, user traffic, and networking events [34]. VINI-enabled network also allowed operators to run multiple protocols on the same underlying physical network, controlling traffic forwarding in individual network devices for each virtual switch independently. From a service provider perspective relying on hardware infrastructure from multiple infrastructure providers, Cabo project in 2007 proposed a separation of infrastructure from services [35]. Cabo promotes the separation of infrastructure from service providers allowing the latter to dynamically update service provisioning for users. Using virtualization and programmable traffic routing, Cabo offers the ability for service providers to run multiple services on networking gear belonging to disparate infrastructure providers. Virtualization in presently deployed networks provides sharing which can be either multiple logical routing devices on a shared platform offering resource slicing such as dedicated memory allocation or traffic forwarding software utilities, all running on the same general purpose computing hardware.

In addition to device virtualization projects, overlay technologies such as Virtual Extensible Local Area Network (VXLAN) were developed as a means to mitigate the limitations of present networking technology allowing extensibility and application in larger data center and cloud deployments [36]. VXLANs utilize MAC-in-IP tunnelling, creating stateless overlay tunnels between endpoint switches, performing encapsulation. Similar to VXLAN is the Network Virtualization using GRE (NVGRE) [37]. NVGRE also embeds MAC-in-IP tunnelling, with a slight difference in the header format. VXLAN packets use UDP-over-IP packet formats sent as unicast between two endpoint switches to assist load balancing, while NVGRE uses the GRE standard header. Another relatively newer virtualization technique is the Stateless Transport Tunnelling (STT) again using MAC-in-IP tunnelling [38]. While the general idea of a virtual network exists in STT, it is, however, enclosed in a more general identifier called a context ID. STT context IDs are 64 bits, allowing for a much larger number of virtual networks and a broader range of service models. STT attempts to achieve performance gains over NVGRE and VXLAN by leveraging the TCP Segmentation Offload (TSO) found in the network interface cards (NICs) of many servers. TSO allows large packets of data to be sent from the server to the NIC in a single send request, thus reducing the overhead. STT, as the name implies, is stateless and packets are unicast between tunnel endpoints, utilizing TCP in the stateless manner (without TCP windowing scheme) associated with TSO. In addition to network virtualization, services such as DNS, access control, firewalls, and caching can be decoupled from the underlying virtual network to solely run as software applications on high volume dedicated hardware and storage. Virtualization of network functionality (VNF) aims at further reducing

the operational and capital expenditure for organizations minimizing dedicated hardware requirements [40].

2.4. Requirement for SDN. While the network virtualization technologies promised greater benefits as compared to conventional and legacy protocols and architectures, the growth in Internet, public and private network infrastructure, and an evolving range of applications required a comprehensive revamping of the existing networking framework. The use of distributed protocols and coordination of changes in conventional networks remains incredibly complex involving the implementation of distributed protocols on the underlying network hardware to facilitate multiple services for traffic routing, switching, guaranteeing quality of service applications, and providing authentication. Keeping track of the state of several network devices and updating policies becomes even more challenging when increasingly sophisticated policies are implemented through a constrained set of low-level configuration commands on commodity networking hardware. This frequently results in misconfigurations as changing traffic conditions requires repeated manual interventions to reconfigure the network; however, the tools available might not be sophisticated enough to provide enough granularity and automation to achieve optimal configurations.

The fundamental requirement of an overall framework which fulfilled a range of operational requirements such as ease of programmability, dynamic deployment, and provisioning while facilitating innovative applications dictated a new networking paradigm capable of satisfying these prerequisites. Some of the technology and operational concerns eventually leading to development of the SDN traffic management framework are detailed as follows:

- (i) Automation: an increased level of automation reducing the overall operational expenditure as well as facilitating effective troubleshooting, reducing unscheduled downtimes, ease of policy enforcement, and provisioning of network resources and corresponding application workloads as required
- (ii) Dynamic resource management: dynamically changing the size of the network and updating the topology and the assigned network resources, which may be further aided by network virtualization
- (iii) Orchestration: orchestrating control of the complete range of network appliances by hundreds or even thousands such as in data centers or larger campus network environments
- (iv) Multitenancy support: with growing proliferation of cloud based services, tenants preferring complete control over their addresses, topology, routing, and security and consequently separating the tenanted infrastructure from hosted services
- (v) Open APIs: users having a full choice of modular plugins, offering abstraction, defining tasks by APIs and not specifically concerned about implementation details. Communication between two nodes to be furnished without the specification of the exact protocol

- (vi) Greater Programmability: a fundamental requirement of present network provisioning being the ability to change device behaviour and configuration in real time according to prevalent traffic conditions
- (vii) Integrated security: the ability to integrate security devices within the network fabric, leading to greater accuracy in detecting security incidents and simplifying management
- (viii) Integrated resource management: in addition to security devices, integrating multiple services seamlessly such as load balancers and resource monitors, which can be provisioned on demand and placed in the network fabric as and when required
- (ix) Improved performance: a control framework offering the ability to incorporate innovative traffic engineering solutions, capacity calculation, load balancing, and a higher level of utilization to reduce carbon footprint
- (x) Network Virtualization: the ability to provision network resources without concerns about the location of individual components such as routers and switches
- (xi) Visibility and real-time monitoring: improving real-time monitoring and connectivity of devices

A centralized view of the distributed network through the SDN control plane provides a more efficient orchestration and automation of network services. While legacy protocols can react after services come online, SDN can foresee additional service requirements and take proactive measures to allocate resources. Furthermore, SDN based network applications deliver highly granular user-defined policies on per-application traffic flows. The following section examines the architecture of the SDN framework in detail.

3. The SDN Architecture

The basic architecture of SDN utilizes modularity based abstractions, quite similar to formal software engineering methods [1, 2]. A typical SDN based network architecture divides processes such as configuration, resource allocation, traffic prioritization, and traffic forwarding in the underlying hardware in three basic layers, namely, application, control, and data planes. Each of the planes has well defined boundaries, a specific role, and also relevant application programmable interfaces (APIs) to communicate with adjacent planes. A comparison between the existing distributed traffic control of individual devices and the centralized SDN architecture is illustrated in Figure 2. The key components of the framework entail the following:

- (i) Data (forwarding) plane: the data plane is a set of network components which can be switches, routers, virtual networking equipment, firewalls, and so forth. The sole purpose of data plane is to forward network traffic as efficiently as possible based on a certain set of forwarding rules instructed by the control plane. SDN architecture makes the networking hardware

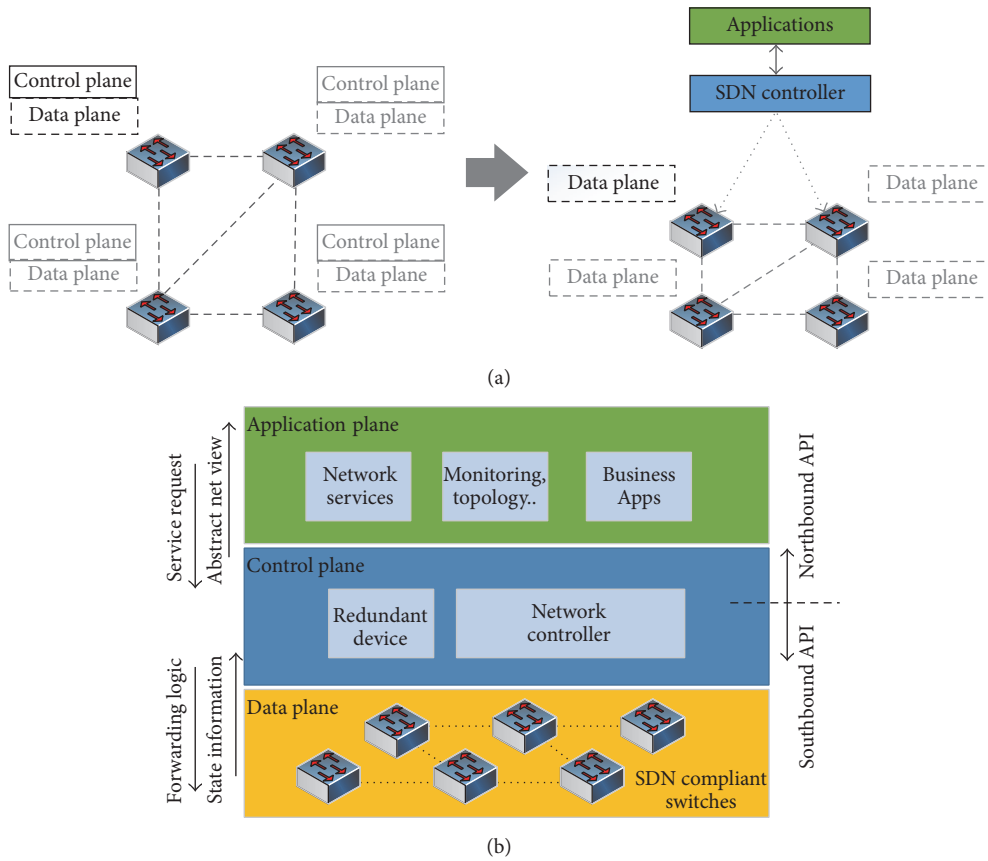


FIGURE 2: Diagram illustrating (a) decentralized and centralized network control and (b) SDN architecture.

rather inoculative by removing forwarding intelligence and isolated configuration per network element and moving these functionalities to the control plane. Communication between data and control planes is achieved by APIs (southbound). At present, the OpenFlow protocol serves as a southbound communication protocol of choice supported by several vendors as well as the ONF.

- (ii) **Control plane:** control plane is responsible for making decisions on how traffic would be routed through the network from one particular node to another based on end user application requirements and communicating resulting network policies to the data plane. The central component of a control plane is the SDN controller. An SDN controller translates individual application requirements and business objectives such as the need for traffic prioritizing, access control, bandwidth management, and QoS into relevant forwarding rules which are communicated to data plane components. Based on the size of the network, there can be more than one SDN controller for additional redundancy [22, 23]. By introducing network programmability through the control plane, it becomes possible to manipulate flow tables in individual elements in real time based on network performance and service requirements. The controller gives

a clear and centralized view of the underlying network giving a powerful network management tool to fine tune performance.

- (iii) **Application plane:** application plane comprises network specific and business applications. An abstract view of the underlying network is presented to applications via northbound APIs. The level of abstraction may include network parameters like delay, throughput, and availability descriptors giving the applications a wider view of the network [2, 7, 20]. Applications in return request connectivity between end nodes and once applications or network services communicate these requirements to the SDN controller, it correspondingly configures individual network elements in the data plane for efficient traffic forwarding.

Centralized management of network elements provides additional leverage to administrators giving them vital statistics of existing network conditions to adapt service quality and customize network topology as needed. For example, during periods of high network utilization, certain bandwidth consuming services like video streaming, large file transfers, and so forth can be load-balanced over dedicated channels. In other scenarios, such as during an emergency (fire alarms, building evacuations, etc.), services such as VoIP can take

control of the network, that is, telephony taking precedence over everything else. Major southbound and northbound communication protocols and prominent controller choices are presented in the following sections.

3.1. Southbound Communication Protocols. Southbound APIs furnish network control enabling the SDN controller to dynamically make network changes as per real-time requirement. The OpenFlow protocol [23] maintained and updated by ONF [5] is the first and most prominent southbound communication interface. OpenFlow defines controller-data plane interaction facilitating administrators to manage traffic according to changing business requirements. Using OpenFlow, flows forwarding constructs can be added and removed in switch flow tables to make the network fabric more responsive to service demands. Besides, the OpenFlow protocol, Cisco OpFlex [41], has gained momentum among southbound APIs. A number of networking vendors have signed up to support implementation of OpenFlow including Juniper, Big Switch, Arista, Brocade, Dell, IBM, NoviFlow, HP, Cisco, Extreme Networks, and NEC, among others. While OpenFlow is quite well-known, it is not the only one available or under development. The extensible messaging and presence protocol (XMPP) [26] has found a certain degree of traction for further deployment especially in hybrid SDN which uses a bulk of protocols such as OSPF, MPLS, BGP, and IS-IS to run on SDN architecture. Operational functionality of OpenFlow, XMPP, and OpFlex is further detailed in the following subsections.

3.1.1. OpenFlow Protocol. OpenFlow is a major southbound API developed in the early stages of SDN paradigm and is meant to communicate control messages between the SDN controller and networking components in the data plane [23]. A typical OpenFlow compatible switch is comprised of one or more flow tables matching incoming flows (and packets) with policy actions such as prioritization, queueing, and packet dropping. The SDN controller can manipulate the flow tables either in (a) real time, reactively, for example, if packet's forwarding path is unknown and a switch sends message to the controller asking for forwarding information or proactively by sending complete flow entries based on requirements dictated by higher applications residing in the application plane. OpenFlow pipeline processing through flow tables is depicted in Figure 3 and a sample table depicting flow table entries is given in Table 2.

Once a packet arrives at the switch, matching is performed in a single flow table and either sent to its destination (outgoing port) or sent to other flow tables as dictated by network control logic instructed by the controller. Matching occurs on the basis of priority in flow table entries with the top matching entry in the flow table and corresponding action executed. If no match is found (called a "table miss"), either the packet is dropped or a request is sent to the controller requesting processing instructions. Packets transverse flow tables in the form of metadata communicated between different tables. Flow entries can also point the packet to particular group actions. Group actions allow a further set of

complex policies to be executed on packets compared to flow tables such as route aggregation and multicasting.

Packets arriving at the ingress port of a switch are generically processed in the following sequence:

- (1) Highest priority matching flow entry in the first flow table is found based on ingress port, metadata, and packet headers. Priority is calculated on a top-to-bottom approach with entries at the top carrying higher priority.
- (2) Relevant instructions are applied which comprise the following:
 - (i) Modify the packet as instructed in the actions list and (or) transmit through an output port.
 - (ii) Update the action set by adding deleting actions in the actions list.
 - (iii) Update metadata.
- (3) Send match data and action set to the next table for further processing.

The fundamental difference between an action list and action set is the time of execution [5]. An action list is executed as soon as packet's data leaves the flow table to make necessary changes to this data, whereas an action set keeps accumulating and is executed once it transverse all relevant flow tables. Flow tables are assigned numbers in sequence and match data (along with action set) can generally only be sent from a table of lower sequence to higher, assuring that packets move in forward direction instead of backward direction in the switch. An OpenFlow compliant switch maintains a TLS control channel with the SDN controller and periodically sends keep alive "hello" messages to communicate state information. To ensure reliability in message delivery between the controller and switch, TCP protocol is used. Well-known TCP ports for OpenFlow traffic are 6633 and 6653 (official IANA port since 2013-07-18). OpenFlow versions have evolved over the past few years offering bug fixes and enhancements. The latest version available at the time of writing is v1.5.

3.1.2. Extensible Messaging and Presence Protocol (XMPP). Extensible messaging and presence protocol (XMPP) was originally designed as a general communications protocol offering messaging and presence information exchange among clients through centralized servers [26]. XMPP remains quite similar to simple message transfer protocol (SMTP); however, the schema is extensible through XML encoding for user customization and additionally the protocol provides near real-time communication. Each XMPP client is identified by an ID which could be as simple as an email address. Client machines set up connections with a central server to advise their presence, which maintains contact addresses and may let other contacts know that a particular client is online. Clients communicate with each other through chat messages which are pushed as opposed to polling used in SMTP/POP emails. The protocol is an IETF standardization of the Jabber protocol and is defined for use with TCP connections in RFC 6121. A number of

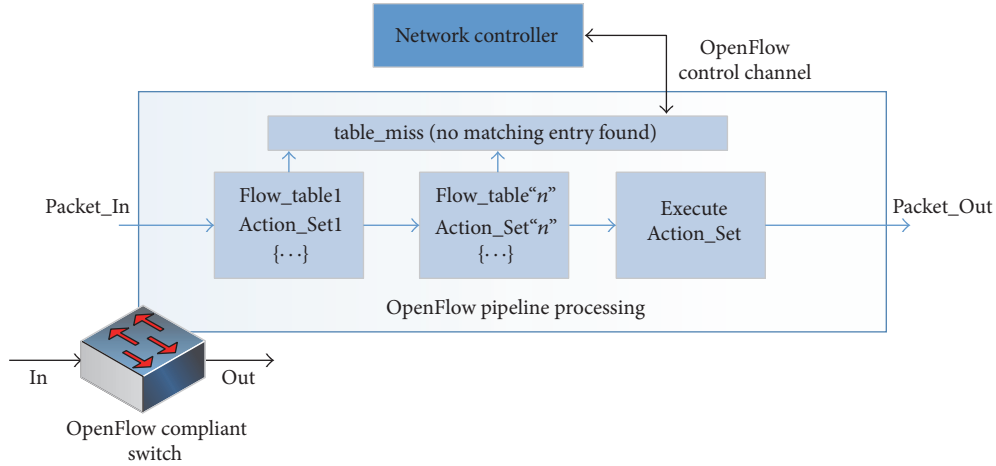


FIGURE 3: OpenFlow pipeline processing.

TABLE 2: OpenFlow Flow Table Entries.

Parameter	Match fields	Counters	Instructions
Functionality	Matching packet headers, ingress ports, instructions and previous table metadata	Update respective counters based on packet matches	Apply per flow actions
Purpose	Traffic segregation for further processing	Measuring flow statistics, real-time traffic monitoring	Flow routing, metering, queueing, QoS, and so forth

open source XMPP implementations are also available with variations being used in programs including Google, Skype, Facebook, and many games. The protocol has found new applications in hybrid SDN, Internet of Things (IoT), and data centers and is being used for managing individual network devices. Network devices run XMPP clients which respond to XMPP messages containing CLI management requests. In data centers, every object such as virtual machine, switch, and hypervisor can have an XMPP client module awaiting instructions from XMPP server for authentication and traffic forwarding as shown in Figure 4. The clients on receiving instructions update their configuration as per server request.

While XMPP is defined in an open standard and follows an open systems development and application approach allowing interoperability among multiple infrastructures, it also suffers a few weaknesses. The protocol by itself does not guarantee QoS of message exchanges between the XMPP client and the server and assured delivery mechanism has to be built on top of XMPP. Additionally, the protocol does not allow end-to-end encryption, a fundamental requirement in modern dispersed and multitenanted network architectures.

Development work is continuing at least to deal with the message delivery, with proposals targeting the creation of a message delivery receipt mechanism.

3.1.3. Cisco OpFlex. Cisco OpFlex is another example of a southbound SDN protocol, facilitating control-data plane communication and aiming to become a standard policy implementing language across physical and virtual environments. In comparison with OpenFlow protocol which centralizes all the network control functions using the SDN

controller, the Cisco OpFlex protocol instead concentrates primarily on implementing and defining the policies [41]. The reason for enhanced focus on policies is to remove the controller scalability and control channel communication from becoming the network bottleneck and pushing some level of intelligence to the devices using legacy protocols. The framework allows policies to be defined within a logical, centralized repository in the SDN controller, and the OpFlex protocol communicates and enforces the respective policies within a subset of distributed policy elements on the switches. The protocol allows bidirectional communication of policies, networking events, and statistical monitoring information. Real-time provision of information may in turn be used to make networking adjustments. The switches contain an OpFlex agent supporting the Cisco OpFlex protocol. Work is also being done on implementing open source OpFlex agent to increase adoption across multiple platforms. Some of the industry giants including Microsoft, IBM, F5, Citrix, and Red Hat have shown commitment to embedding OpFlex agent in their product line. OpFlex relies on traditional and distributed network control protocols to push commands to the embedded agents in switches. One of the main reasons for the early adaption of OpenFlow has been the level of control it can offer to developers for designing network control applications with minimal support from network vendors. To standardize OpFlex, Cisco submitted the protocol to IETF standardization process and several vendors are presently working to standardize as well as increase the adoption of the protocol.

3.2. Northbound Communication Protocols. Since the inception of SDN, a number of networking vendors have started

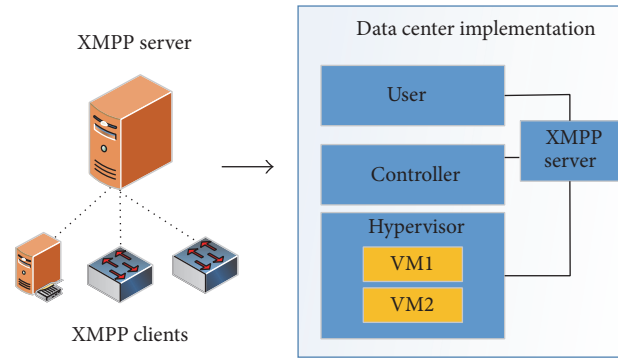


FIGURE 4: XMPP client-server communication.

actively developing SDN oriented applications with the aim of expediting adoption and reducing the OPEX and CAPEX of future IT network infrastructures. The applications themselves vary in scope with some providing a comprehensive network monitoring and control solution while others solely targeting a particular aspect of load balancing, security, and traffic optimization through SDN controllers. The architecture and APIs (northbound) of SDN applications vary between vendors. Some have incorporated SDN controllers inside applications, while others have defined custom northbound APIs for policy translation between controllers and their own higher application layer SDN services. As per the ONF SDN framework [1], applications might act as an SDN controller in their own right or liaise with one or more SDN controllers to gain exclusive control of resources exposed by controllers. Applications can exist at any level of abstraction with a general perception that the further north (higher) we move in SDN framework, the greater the level of abstraction. A specific distinction between applications and controller is not precise [2, 3]. A controller-application interface may mean different things to different vendors. However, the fundamental principle of abstracting network resource view for use by applications and allowing real-time network programmability forms the cornerstone of SDN control plane.

The ONF constituted a special working group in June 2013 towards standardizing architecture across the industry for northbound interfaces (NBIs) [5]. Although there is considerable debate within industry whether such a standardized interface is even required, the benefits of having an open northbound API are also significant. Open northbound API allows developers from different areas of industry and research to develop a network application, as opposed to only equipment vendors. It also gives network operators the ability to quickly modify or customize their network control. The ONF, the consortium, has therefore so far avoided northbound API standardization to allow maximum innovation and experimentation. As a direct result, more than 20 different SDN controllers that are currently available feature varying northbound APIs, based on the needs of applications and orchestration systems residing above. Despite the ONF's efforts, there is a chance there will not ever be a standardized northbound API. Routing and switching

vendors that traditionally rely on network-based applications and features to differentiate their hardware are positioning themselves to maintain profitability in the SDN arena. These vendors will invest in custom software, and OpenFlow will run concurrently to a native operating system and complement the existing control plane. The result is a complex and crowded ecosystem. The following subsections review the two popular northbound APIs, the RESTful [42] and Java based OSGi [43], interface prevalent in SDN controllers.

3.2.1. Representational State Transfer (RESTful). Representational state transfer or simply REST follows the software architecture style developed for World Wide Web consortium (W3C) encompassing all client-server communications. The concept was originally introduced by Fielding in [42]. The main goals of the scheme are to offer scalability, generality, and independence and allow the inclusion of intermediate components between clients and servers to facilitate these necessary functionalities. Both clients and servers can be developed independently or in tandem; there is no particular necessity to have both developed by same vendor. A schematic diagram representing RESTful calls is shown in Figure 5. The server component is stateless and clients keep track of their individual states to allow scalability. Server responses can be cached for a specified time. Every entity or global resource can be identified with global identifiers such as a URI and is able to respond to create, read, update, and delete (CRUD) operations. The uniform interface for each resource is GET (read), POST (Insert), PUT (write), and DELETE (remove). Data types can define network components such as controller, firewall rule, topology, configuration, switch, port, link, and even hardware. RESTful is prevalent in most controller architectures as the northbound interface of choice along with Java APIs. One of the major drawbacks of RESTful, however, is the lack of public subscription or live feed informing the application/service of network changes. Like HTTP, REST does not tell when a page has changed and requires frequent refresh. Application developers, therefore, periodically use loop calls to retrieve and subsequently post updates to individual switches based on predefined policies. REST support is included in almost all major SDN controllers including Ryu [44] and OpenDaylight [45] as well as several vendor proprietary platforms.

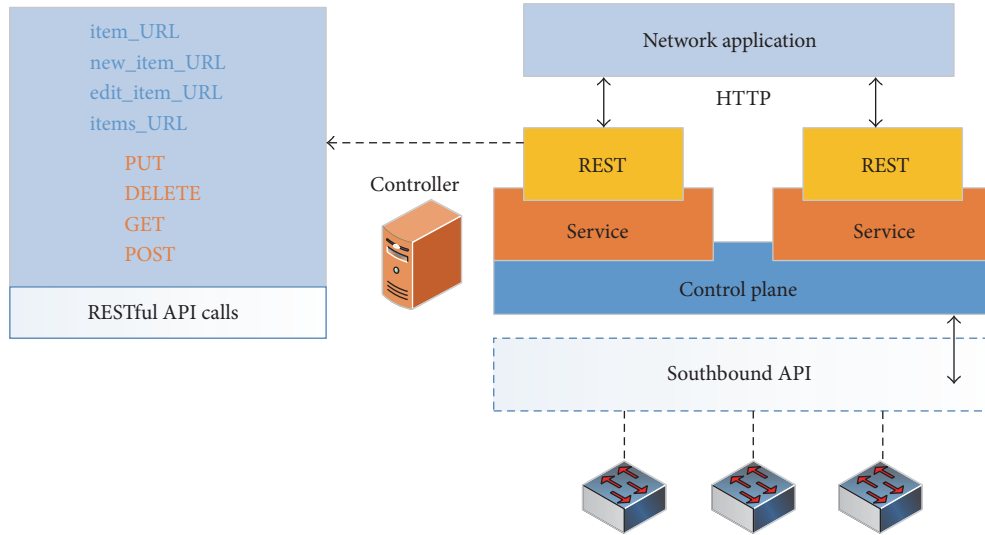


FIGURE 5: RESTful application programming interface (API).

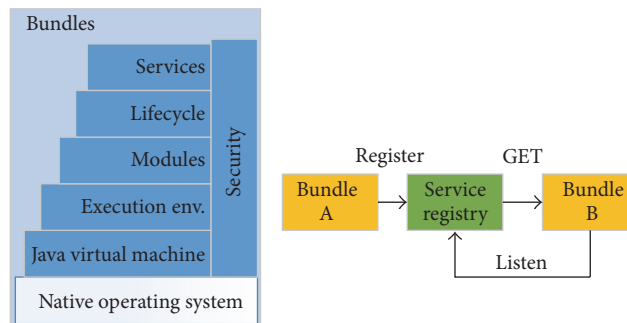


FIGURE 6: Open Services Gateway initiative (OSGi).

3.2.2. Open Services Gateway Initiative (OSGi). Open Services Gateway initiative (OSGi) is a set of specifications for dynamic application composition using reusable Java components called bundles [43]. Bundles publish their services with OSGi services registry and can find/use services of other bundles as shown in Figure 6. Bundles can be installed, started, stopped, updated, and uninstalled using a lifecycle API. Modules define how a bundle can import/export code. Security layer handles security and execution environment defines what methods and classes are available in specific platform. A bundle can get service or it can listen for a service to appear or disappear. Each service has properties that allow others to select among multiple bundles offering the same service. Services are dynamic and a bundle can decide to withdraw its service which will cause other bundles to stop using it. Bundles can be installed and uninstalled on the fly. The OpenDaylight project [45] is one major example of a SDN controller platform built using the Java based OSGi framework. OSGi allows the starting, stopping, loading, and unloading of Java based network (module) functionalities. In comparison, platforms, such as Ryu [44], do not offer OSGi support and the controller has to be stopped and restarted with the needed modules or a custom REST method is built

with all required functionalities included to avoid controller restarts. A few other SDN platforms supporting OSGi include Beacon [46], Floodlight [47], and ONOS [48].

3.2.3. Model-Driven Service Abstraction Layer (MD-SAL). The OSGi framework described above forms the back-end for ODL controller [45] allowing dynamic loading and binding of bundles (JAR files) for exchanging information. Application containers built on top the OSGi framework, such as Karaf [49], simplify the operational aspects of packaging and installing of applications in ODL. To further facilitate application development, model-driven approach to service abstraction used in the ODL controller [50] offers developers and network administrators an opportunity to unify the northbound and southbound APIs with the data structures of multiple services and individual components of the controller [45]. The data structure itself is described by Yet Another Next Generation (YANG) language used to model the service and data abstractions as a single system [51]. YANG (defined in RFC 6020) models semantics and data organization including configuration or operation data as a tree. The controller northbound API can utilize the self-describing data in, for example, XML (YANG model), simplifying the development

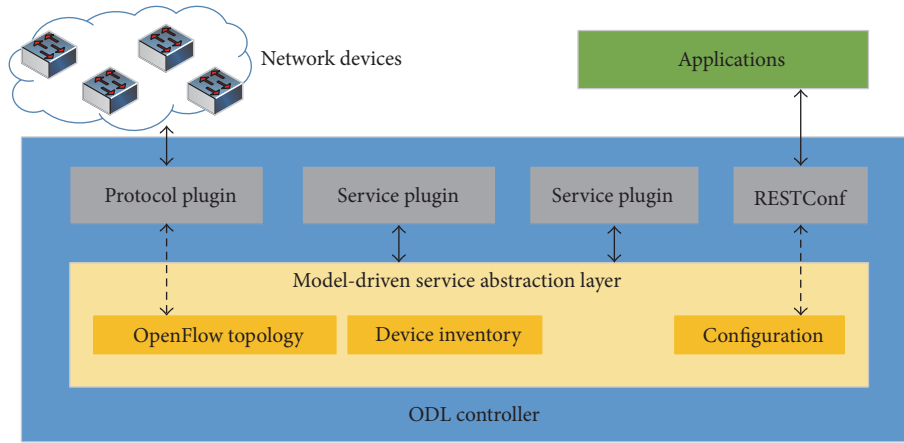


FIGURE 7: Model-Driven Service Abstraction Layer (MD-SAL) architecture.

of additional controller-driven functionalities as well as SDN applications. A schematic representing MD-SAL is provided in Figure 7.

Modules providing specific network functionality can define a schema, allowing simple interpretation of data structures through the service abstraction layer (SAL). MD-SAL uses APIs to connect and bind requests and services and provides an extra layer containing all the necessary logic to receive and delegate requests [52]. The SAL architecture itself comprises (i) top-level subsystem consisting of controller components or applications that use the controller SAL to communicate with other controller components and plugins (data store, validator, etc.) and (ii) nested subsystems that expose a set of functionality and may have multiple instances attached to the overall system (network elements, virtual systems, etc.). The MD-SAL approach is relatively agnostic supporting any service model. Furthermore, the scheme stitches together horizontal modules and allows developers to use generic interfaces for service discovery and subsequent utilization.

3.2.4. Intent-Based Approach. Although the SDN paradigm allows for a high level of abstraction which may not be available in legacy network settings, such an abstraction is increasingly useful when the northbound interface (NBI) allows SDN applications to dictate their *intent* without necessarily having to specify the *methods* to achieve it. As opposed to providing low-level rule-based configurations, a typical intent-based approach allows a user or application to only specify the *intent* such as the requirement of low latency path between two nodes, bandwidth reservation, and real-time monitoring for re-evaluation of selected paths. The SDN controller in turn translates these *intents* into low-level configuration commands in the data plane for subsequent execution. The application or user, therefore, is oblivious to the underlying infrastructure configuration providing added flexibility and automation for application developers and enabling agile deployment of services [53]. Intent-based NBI represents an evolution from static to dynamic network setup through simple application *intent* specification, programmed

by the controller into physical and virtual devices. An illustration of an intent-based NBI is depicted in Figure 8. Intent-based descriptors make a compelling case for implementation in SDN because of a number of benefits such an approach brings from both an operations and architecture perspective, some of which are briefly detailed as follows [53–55]:

- (i) *Application scalability*: an application developer does not require prerequisite knowledge about the actual network, focusing on developing the application rather than worrying the workings of the application with the infrastructure.
- (ii) *Greater portability*: IBN allows an application developed for one SDN environment to be easily ported to another without redesigning and is therefore, agnostic to individual (controller) vendors.
- (iii) *Increased coherence*: decoding low-level changes caused by SDN applications running simultaneously can sometimes lead to resource conflict, leaving the controller unable to understand the user or application intent. Intent-based directives to the SDN controller aim to avoid control-splitting, translating multiple application intent into cohesive device configurations.
- (iv) *Contextual management*: managing network resources at a lower level of abstraction for multiple services is quite difficult and prone to errors. Intent-oriented descriptions make it possible for administrators to not only provide conflict resolution but also define a cumulative intent of multiple application during service initiation, allowing ease of managing the SDN.

To permit practical deployment, intent NBI, however, requires a language for translating user or application intent as highlighted in [9, 54, 56, 57]. Industry vendors have been working on developing intent-based NBI for their proprietary controllers [58, 59]. The ONOS [48] also provides application intent at the NBI, although intent is described in the format of a network policy, allowing applications to

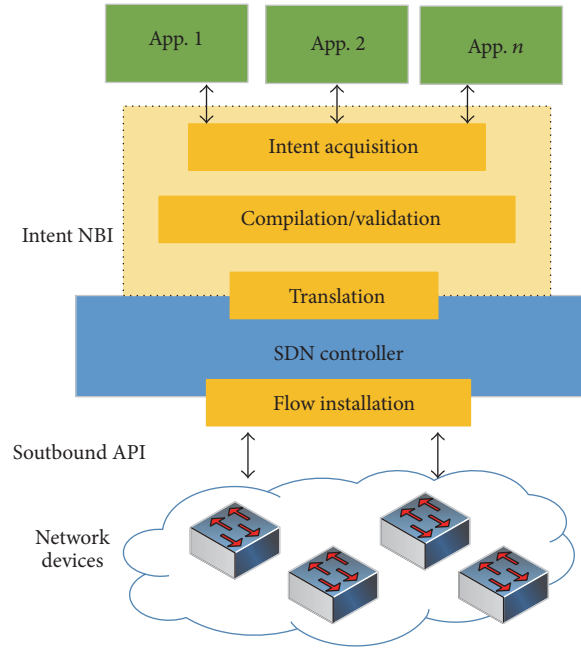


FIGURE 8: Intent-based northbound interface illustration.

specify policies which are compiled and installed as device flow rules. In terms of community efforts, the ONF NBI Working Group has been working on specifying the information and architectural model for an intent-based interface to the SDN controller [55]. Other projects especially the ODL Network Intent Composition (NIC) initiative also aims to aid administrators in managing and directing network services by describing the intent for network behaviour through a northbound interface, facilitating abstracted policy semantics instead of specifying data plane (flow) rules [60, 61]. The project uses existing ODL functions and southbound plugins and is designed to be protocol agnostic, that is, able to use any control protocol. The ONF NBI Working Group and NIC initiative comprise a diverse set of projects, operators, and vendors collaborating with the open source community to bring intent-based approach to the SDN NBI.

3.3. Network Controllers and Switches. The SDN controller maintains and applies network policies required by higher application and services as well as translating and configuring these policies in individual network devices. As mentioned earlier, once a packet arrives at switch, in case of a table miss (absence of flow entry), it may get forwarded to the controller which determines the next course of action for the respective traffic flow. Computed instructions such as adding, removing, and modifying flow entries are carried out in switches using a southbound communication interface (e.g., OpenFlow). The controller in essence centralizes the network intelligence. The controller is assigned a configurable IP address and individual switches communicate with the controller using a predetermined port number using standard TLS or TCP connection. The control channel between the controller and switches is independent of the traffic forwarding framework. Applications and services requiring communication between two

endpoints may communicate these to the controller using the northbound interface, and the controller translates these into low-level configuration commands in individual switches. A schematic representing generic controller architecture is given in Figure 9. Depending on redundancy requirements, switches may communicate with either a single or several controllers [62]. Intercontroller communication is usually served by an external legacy protocol such as the Border Gateway Protocol (BGP) or the Session Initiation Protocol (SIP) over TCP channels to exchange routing information. Multiple controllers can improve the reliability of the system. In case of failure of one controller or control channel, the switch can obtain flow forwarding instructions from another controller instance. The number of controllers and their placement, however, greatly depends on the topology and operational requirements of an organization. Two popular schemes proposed include the vertical approach where multiple controllers are in effect managed by controller(s) at a higher layer and the horizontal approach in which controllers establish a peer-to-peer communication relationship [63–65].

Controllers are usually hosted on network attached server. In case of multiple controllers, OpenFlow dictates that the switches maintain a control channel with each controller. A summary of commonly used OpenFlow compliant controllers is given in Table 3, along with their development platform. The two categories of controllers covered in the table are general and special purpose controllers. NOX and POX are examples of early stage general purpose controller platforms during SDN evolution [66, 67]; however, POX remains relevant offering OpenFlow support as well as a visual topology manager. Ryu by NTT Corporation developed in Python programming language has found increased applicability in several research studies, being a complete SDN ecosystem

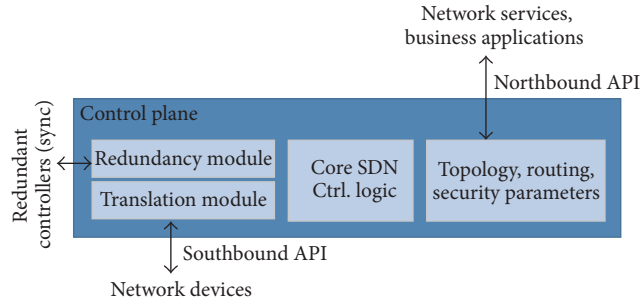


FIGURE 9: SDN controller schematic.

supporting the OpenFlow protocol as well as the RESTful at the northbound interface [44]. The OpenDaylight (ODL) controller platform founded and led by several industry giants offers Java based development and deployment of carrier-grade SDN solutions [45]. Special purpose controllers such as FlowVisor [68], RouteFlow [69], and Oflops [70] serve specific tasks including transparent proxy between switches and multiple controllers, virtualized IP routing over OpenFlow network switches, and benchmarking switch performance in addition to serving as network controllers. Furthermore, platforms such as the open network operating system (ONOS) provide an operating system resilient enough for carrier-grade deployment of software defined networks [48]. The ONOS GUI provides a multilayer view of the underlying network and allows operators to peruse network devices, links, and errors with subsequent policy implementation.

In addition to the several network controllers on offer, presently there are several types of SDN software and hardware switches available. The software switches can be used to run SDN test simulations as well as develop protocols and services. Open vSwitch, for example, is now part of the Linux kernel (as of version 3.3) and facilitates both the ability to serve as virtual gateway between physical and virtual services as well as a testing platform to be used in tandem with SDN topology simulation tools such as Mininet. In addition to software switches, industry giants such as IBM, HP, and NEC have also brought physical carrier-grade switches to market. A summary of current OpenFlow switch implementations are presented in Table 4, along with their brief description and development platform (language) where available. The networking industry has taken interest in SDN evidenced by the availability of commercial hardware switches which are OpenFlow enabled.

4. Simulation, Development, and Debugging Tools

The development of SDN has seen the advent of several key simulation and emulation test beds to carry out feasibility studies and introduce new protocols and services. The set of tools available for the purpose can be broadly divided into four categories: (i) simulation and emulation platforms, (ii) software switch implementations, (iii) white box solutions, and (iv) debugging and troubleshooting tools. A summary

description of major utilities within each category and their description are given in Table 5. An overview of tools is given below.

4.1. Simulation and Development Platforms

4.1.1. Mininet. Among the emulation tools, Mininet [96] is the most prominent. The platform allows an entire network based on OpenFlow to be emulated over a single or a cluster of machines. The distribution of Mininet nodes and links over a cluster of machines utilizes the resource of each machine, adding scalability to emulate larger networks requiring more computation and communication bandwidth than available on a single Mininet server. Mininet simplifies the development and deployment of new services by providing a software platform to create virtual machines, hosts, and network switches connected to an in-built (ovs-reference) or user-defined controller for testing purposes. The latest Mininet v2.2.2 supports OpenFlow versions up to 1.3 (along with Open vSwitch v2.3) by default and can also be customized to use an external user space switch such as the softswitch13 [84].

4.1.2. NS-3. The network simulator has long been used by the networking community to test and develop networking protocols and services. The latest ns-3 simulator offers support for OpenFlow switches; however, it is limited to a very early version of OpenFlow v0.89 [97]. While official work is continuing on introducing newer updated versions of OpenFlow, a specialist OpenFlow 1.3 module for ns-3, namely, OFSwitch13, module has been designed externally [91]. The module relies on the ofsoftswitch13 library providing a data path (switch) implementation in the user space and to convert OpenFlow v1.3 messages from wire format.

4.1.3. OMNeT++. The OMNeT++ is a discrete event simulator allowing the development and testing of SDN based models [98]. SDN oriented projects can be integrated with OMNeT++ using an OpenFlow components and an INET Framework.

4.2. Software Switch Implementations. A nonexhaustive summary of software switches which are also used for experimentation and new service development are given in Table 5.

TABLE 3: Popular OpenFlow compliant controller implementations.

Controller	Implementation	Open source	Developer	Description
NOX	C++/Python	Yes	Nicira	The first OpenFlow controller [66].
POX	Python	Yes	Nicira	Controller supporting OpenFlow having a high-level API including topology graph and virtualization support [67].
Ryu	Python	Yes	NTT, OSRG	Network Operating System (NOS) that supports OpenFlow [44].
OpenDaylight	Java	Yes	Industry consortia	Platform for building programmable, software defined network applications [45].
Beacon	Java	Yes	Stanford University	Java based controller that supports both event-based and threaded operations [46].
Floodlight	Java	Yes	Big Switch	OpenFlow controller, forked from the Beacon controller [47].
Helios		No	NEC	Controller providing shell environment for integrating experiments [71].
Trema	C/Ruby	Yes	NEC	Full-stack framework for developing OpenFlow controllers in Ruby and C [72].
Jaxon	Java	Yes	Independent	NOX-dependent Java based OpenFlow controller [73].
MUL	C	Yes	Kulcloud	OpenFlow controller having multithreaded infrastructure at its core and designed for performance and reliability in mission-critical environments [74].
IRIS	Java	Yes	IRIS Team-ETRI	OpenFlow Controller having horizontal scalability for carrier-grade network, high availability and multidomain support [75].
Maestro	Java	Yes	Rice University	OpenFlow operating system for orchestrating network control applications [76].
NodeFlow	JavaScript	Yes	Independent	OpenFlow controller written in pure JavaScript [77].
NDDI - OESS	C++	Yes	Internet2, Indiana University	Application to configure and control OpenFlow enabled switches through a simple and user friendly interface [78].
RouteFlow	C++	Yes	CPqD	<i>Special purpose</i> provides virtualized IP routing composed of an OpenFlow controller application, an independent server and physical network emulation [69].
FlowVisor	Java	Yes	Stanford University/Nicira	<i>Special purpose</i> OpenFlow controller, a transparent proxy between switches and multiple controllers [68].
SNAC	C++	No	Nicira	<i>Special purpose</i> controller built on NOX uses a web-based policy manager [79].
Resonance	NOX+OpenFlow	Yes	Georgia Tech.	<i>Special purpose</i> access control application built using NOX [66] and OpenFlow [23].
Oflaps	C	Yes	Cambridge, Berlin, Big Switch	<i>Special purpose</i> standalone controller used to benchmark performance and test an OpenFlow switch [47].
ONOS	Java	Yes		Open source scalable control plane cluster offering GUI and OpenFlow as well as NETCONF support [48].
ovs-controller	C	Yes	Independent	Reference controller packaged with Open vSwitch [80].

Prominent cases such as the Open vSwitch have been implemented in multiple platforms including Mininet and ns-3. A brief overview of some of the prominent software switches available is given below.

4.2.1. Open vSwitch. The Open vSwitch is one of the most widely deployed software switches. It employs an OpenFlow stack that both can be used as a virtual switch in

virtualized network topologies and has also been ported to multiple hardware/commodity switch platforms [80]. The Open vSwitch is a part of the Linux kernel since version 3.3.

4.2.2. ofsoftswitch13. The ofsoftswitch13 running in the user space also provides support for multiple OpenFlow versions [84]. The soft switch supports Data Path Control (Dpctl), a management utility to directly control the OpenFlow switch,

TABLE 4: Common OpenFlow Compliant Switches and Standalone Stacks.

Switch	Implementation	Category	Description
Open vSwitch	C/Python	Software	OpenFlow stack that is used both as a virtual switch and ported to multiple hardware platforms [80].
Indigo	C	Software	Software running on hardware switching implementations and based on the Stanford reference [81].
OpenFlowJ	Java	Software	OpenFlow stack written in Java [82].
OpenFaucet	Python	Software	Python implementation of the OpenFlow 1.0 protocol [83].
ofsoftswitch13	C/C++	Software	User space software switches implementation [84].
Pantou	C	Software	OpenFlow port to the OpenWRT wireless environment [85].
Oflib-node	JavaScript	Software	OpenFlow protocol library for Node.js converting between the protocol messages and JavaScript objects [86].
OpenFlow Reference	C	Software	Minimal OpenFlow reference stack that tracks the specification [22].
Pica8	C	Physical and software	Software platform for hardware switching chips which includes L2/L3 stack [87].
A10 Networks—AX Series	Proprietary	Physical and software	Physical and software appliances (AX Series), offering L4-7 programming [88].
Big Switch Networks - Big Virtual Switch	Proprietary	Physical and software	Data center network virtualization application built upon an OpenFlow switches [89].
Brocade ADX Series	Proprietary	Physical and software	Secure and scalable application service infrastructures using the RESTful API on northbound interface [90].
NEC ProgrammableFlow Switch Series	Proprietary	Physical and software	Series offers network virtualization, multipath routing, security, and programmability [91].
ADVA Optical - FSP 150 & 3000	Proprietary	Physical	FSP 150 carrier Ethernet and FSP 3000 transport layer products [92].
IBM RackSwitch G8264	Proprietary	Physical	Offers low cost flexible connectivity for high-speed server and storage devices in DC environments [93].
HP 2920, 3500, 3800, 5400 series	Proprietary	Physical	Advanced modular switch series built on programmable ASICs offering scalable QoS and security [94].
Juniper Junos MX, EX, QFX Series	Proprietary	Physical	Series supports different versions of OpenFlow varying with model [95].

allowing the addition and deletion of flows, query switch statistics, and modify flow table configurations. Although ofsoftswitch13 supports a variety of OpenFlow features, it has recently run into some compatibility issues with latest versions of Linux (Ubuntu 14.0 and beyond) and developer support has also stagnated.

4.2.3. Indigo. The Indigo project is an open source implementation of OpenFlow which can be run on a range of physical switches and utilizes the hardware features of existing Ethernet switch ASICs to run OpenFlow pipeline at line rates [81]. The implementation is based on the original OpenFlow reference implementation from Stanford and currently supports all features required in the OpenFlow 1.0 standard.

4.2.4. Pica8 PicOS. The PicOS by Pica8 is a network operating system allowing network administrators to build flexible and programmable networks using white box switches with OpenFlow [87]. The proprietary software allows the integration of OpenFlow rules to be used in legacy layer 2/layer 3

networks, without disrupting existing network and creating a new one from scratch.

4.2.5. Pantou. Pantou modifies a commercial wireless router and access point to an OpenFlow enabled switch. The OpenFlow protocol is implemented as an application on top of OpenWRT platform [85]. The OpenWRT platform used is based on the BackFire release (Linux v2.6.32), while the OpenFlow module is based on the Stanford reference implementation in user space.

4.3. White Box Switch Implementations. White box switches allow network operators to use off-the-shelf switching hardware in the SDN data plane. Being the foundation building blocks of a network, the ability to use generic switching hardware allows organizations to leverage the benefits of individual components suited for specific SDN applications as opposed to investing in relatively costly all-in-one vendor solutions. The white box itself may have a preinstalled operating system or consist of a bare metal device, leaving it to the operator to select and load software which would

TABLE 5: Common Simulation Platforms and Debugging Tools.

Category	Purpose	Software and tools
Emulation and simulation	Emulating network topologies as well as providing a reference for network event simulation	Mininet [107], ns-3 [108], OMNeT++ [109]
Software switches and platforms	A software platform to test and validate switch-controller behaviour and southbound protocol working	Open vSwitch [80], ofsoftswitch13 [84], Indigo [81], Pica8 PicOS [87], Pantou [85]
Debugging and troubleshooting	Specialist tool set to debug SDN behaviour at the switch and controller level	STS [110], Open vSwitch [80], NICE [111], OFTest [112], Anteatr [107], VeriFlow [108], OFRewind [113], NDB [109], Wireshark [114]

aid in integrating the product into a larger SDN ecosystem to facilitate networking features. Open source Linux-based platforms and other specialist operating systems such as ONOS [48] aid administrators in customizing the white box devices according to applications running above the SDN controller [99, 100]. The OpenFlow (or other southbound protocol) may be used to program the flow forwarding tables in white box switches for route construction and fulfil other application requirements. Having a flexible management framework, white boxes can also support horizontal and vertical integration with additional open source tools such as OpenStack [99, 100], Puppet [101], Chef [102], and CFEngine [103], simplifying network operations in multiple environments ranging from SDN enabled cloud infrastructures to enterprise data centers. Increasing interest in open hardware platforms has also resulted in collaborative networking community efforts such as the Open Compute Project (OCP) which proposes the reimagining of network hardware to make it more efficient, flexible, and highly scalable [104].

In addition to reducing CAPEX, the use of SDN tools in combination with off-the-shelf hardware in data centers allows reduction in time for provisioning new services. At the access level, white box switches may be utilized to offer functionalities from wireless network control to LAN traffic forwarding, orchestrated by the SDN controller. Customizable feature adoption allows administrators to utilize the white box devices in multiple settings as dictated by the SDN applications. White box solutions, however, require a significant deal of expertise from administrators to accurately configure the devices for subsequent use, sometimes without sophisticated manufacturer after-sales support. A number of early deployments of white box switches are therefore seen in relatively large service providers, leveraging existing network management experience to offer greater innovation using SDN programmability with commodity hardware [105]. Open source initiatives such as the Open Network Install Environment (ONIE) further enables administrators to install any network operating system on hardware device

with a great deal of automation, essentially enabling management of white box switches which is quite similar to servers [106].

Branded white box switching gear without a default operating system is available from a number of manufacturers. The software to be loaded (operating system) is available from either relatively new start-up companies to more established network solution providers. A nonexhaustive list of white box hardware and operating system software is summarized in Table 6. To differentiate from bespoke legacy solutions, vendors have also created new product lines and distribution channels for white box/bare metal switches, specifically meeting open network deployment requirements [115–117]. The resulting port speeds may range from the basic 1 Gbps all the way up to 100 Gbps. Although the impact of white boxes in data centers and enterprise networks may result in more innovation and flexibility, vendors might deemphasize the overall influence, noting that white box solutions may extract value only for organizations with tremendous prior networking expertise [118]. Lack of support options, specific features (protocols), and troubleshooting assistance from vendors may deter smaller concerns from adopting white box switching solutions despite the benefits underlined above. Proponents of white box switching in SDN, however, argue that integrated switch models have run their course and bare metal solutions offer speeds equivalent to branded legacy switches [119]. Furthermore, the use of white box switching in SDN is also potentially significant in lowering operating costs. Switching devices which can be run and managed as servers allows operators to tightly integrate SDN applications with the underlying network fabric, a step up from somewhat expensive vendor-locked solutions.

4.4. Debugging and Troubleshooting Tools. Debugging and troubleshooting tools serve as vital resources for development and testing of SDN based services. The following list presents some of the popular SDN debugging and verification tools.

4.4.1. SDN Troubleshooting System (STS). STS simulates the network devices of your network while also allowing enough programmatically to generate and examine various test case deployments [110]. Users can interactively visualize the network states, the real-time changes, and also automatically determine the events that trigger deviant behaviour and identify bugs. The implementation is based on the POX controller platform, with feasibility to use other OpenFlow compliant controllers supporting OpenFlow v1.0.

4.4.2. Open vSwitch Specific Tools. The Open vSwitch comes with a comprehensive set of tools to debug the switch and network behaviour. The utilities are comprised of the following:

- (i) ovs-vsctl: used for configuring the switch (daemon) configuration database (known as ovs-db.).
- (ii) ovs-ofctl: a command line tool for monitoring and administering OpenFlow switches.

TABLE 6: White box switching solutions.

Manufacturer	Solution	Brief description
Accton/Edgecore	Switch	Bare metal hardware with a choice of independent open software for NOS [115]
Delta Networks/Agema Systems	Switch	SDN capable switches supporting ONIE installer and OpenFlow [116]
Quanta Cloud/iwNetworks	Switch	Top-of-Rack (ToR) or spine switches for high performance data centers, loaded with ONIE [117]
Interface Masters	Switch	Niagara switch series, offering modular switch configurations conforming to open network deployment (OCP) [120]
Alpha Networks	Switch	Bare metal ToR switches, with ONIE support [121]
Penguin Computing	Switch	Arctica switch line, offered as an open solution, with a choice of preloaded Cumulus Linux operating system [122]
BigSwitch	OS	Switch Light NOS, Linux-based switching software for programming white box switches and serving as an agent for applications running above SDN controller [123]
Cumulus Networks	OS	Cumulus Linux provides standard networking functions such as bridging, routing, access control, and VXLAN overlays [124]
Gigamon	OS	GigaVUE-OS, built on a Linux kernel, the OS is available on white box hardware providing flow management capabilities [125]
Broadcom	OS	ICOS/FASTPATH runs as a Linux application with seamless integration of Linux tools while offering server like experience [126]
Dell	OS	OS10 software framework that facilitates customization supporting the open networking model specifically suited to SDN data centers [127]
IP Infusion	OS	OcNOS, a modular, multitasking Linux-based operating system, offering integration with commodity network hardware [128]

(iii) `ovs-dpctl`: used to administer Open vSwitch data paths (switches). In addition to Open vSwitch `ovs-dpctl`, a reference `dpctl` comes with the OpenFlow reference distribution and enables visibility and control over a single switch's flow table. The syntax of commands used by the utilities is quite different.

(iv) `ovs-appctl`: used for querying and controlling Open vSwitch daemons.

4.4.3. NICE. NICE offers an automated testing tool used to identify and check bugs in OpenFlow programs [111]. The tool applies model checking to explore entire state of the controller, the switches, and the hosts. To address scalability issues, the tool uses model checking with symbolic execution of event handlers (identifying the representative packets that exercise code paths on the controller). NICE prototype tests Python applications using the NOX platform.

4.4.4. OFTest. OFTest is an OpenFlow switch test framework built-in Python and includes a collection of test cases [112]. The tool is based on unittest which is included in the standard Python distribution. OFTest hosts the switch under test and the OFTest code runs on the test switch. Both control plane and data plane side of switch connections can be tested by sending and receiving packets to the switch as well as polling switch counters.

4.4.5. Anteater. Anteater attempts to check network invariants that exist in the networking devices, such as connectivity

or consistency [107]. The main benefit of using Anteater is that it is agnostic to protocols and will catch errors that result from faulty firmware as well as control channel communication.

4.4.6. VeriFlow. VeriFlow allows real-time verification and resides between the controller and the data plane elements (switches) [108]. The framework allows pruning of flow rules that may result in anomalous network behaviour.

4.4.7. OFRewind. OFRewind is another tool that allows debugging of network events in both the control and data plane and to log these at different levels of detail to be replayed later in examining problematic scenarios and localize troubleshooting efforts [113].

4.4.8. Network Debugger (NDB). NDB implements traffic breakpoints and packet-backtraces for an SDN environment [109]. Similar to the popular software debugging utility gdb, users can isolate networking events that may have led to an error during traffic forwarding. It works using the OpenFlow API to configure switches and generate debugging events. NDB then acts as a proxy intercepting OpenFlow messages between switches and the controller. The debugger relies on OpenFaucet Python module implementing OpenFlow v1.0.

4.4.9. Wireshark. The popular network analyser Wireshark can be deployed on the controller or Mininet host to view OpenFlow exchange messages between the controller and individual switches. The OpenFlow dissector is available

in the current Wireshark release [114]. OpenFlow control packets can be directly filtered while capturing using the TCP control channel traffic ports (6633 and 6653). The captured packets provide a useful learning tool to understand switch-controller behaviour.

5. SDN Applications

A number of networking vendors have started actively developing SDN applications with the aim of expediting adoption to reduce the total time to market and total cost of ownership of future IT network infrastructures. SDN applications vary in their scope with some providing a comprehensive network monitoring and control solution to others solely targeting a particular aspect of load balancing, security, and traffic optimization through SDN controllers. As per the ONF SDN framework [1], applications might act as an SDN controller in their own right or liaise with one or more SDN controllers to gain exclusive control of resources exposed by controllers.

Software defined networking has found a great deal of applicability in wide range of networking avenues. Real-time programmability through the centralized controller has presented opportunities in data center networking and large campus environments as well as experimental designs focusing on enhancing end user experience and making residential networks more manageable. Furthermore, mobile operators have also shown keen enthusiasm in bringing the technology to 5G/LTE mobile networks to allow simplified yet rapid development and deployment of new services. Some of the key applications of SDN are highlighted as follows with a summarization given in Table 7.

5.1. Wireless Communication. Due to the real-time programmability and potential to seamlessly introduce new services and applications to consumers, the SDN paradigm has been ported to mobile communication networks. A programmable wireless data plane offering flexible physical and MAC address based routing, in comparison to layer 3 logical address based traffic forwarding, has allowed developers to fine tune mobile communications performance [129, 130]. Using the control plane, user traffic can be segregated and routed over different protocols such as WiMAX, 3GPP, and advanced LTE. The applications of SDN to wireless network environments are discussed as follows.

5.1.1. Mobile Cellular Communication. There have been growing efforts to include the SDN layers in upcoming 5G mobile communications realm and move from a flat topology which increasingly relies on the core to a modular framework. Current cellular technologies are relatively inflexible coupled by limitations in link capacities, making real-time service provisioning difficult and prone to errors. The redesigning of cellular networks using SDN principles adds modularity to the existing infrastructure, with each layer encapsulating horizontally chained protocol stacks orchestrated by a network operating system residing at the top [131]. A cellular SDN framework (CSDN) leveraging network function virtualization (NFV) allows optimized control through contextual analyses of user data to create intelligent traffic

forwarding policies [132]. Information from mobile edge networks is gathered and used by the SDN control plane to implement real-time end-to-end routing policies and enable service innovation which greatly enhances user experience as opposed to conventional mobile networking. Increasing use of virtual machines (NFV) to support network services also facilitates service chaining and efficient management of flow using a centralized SDN controller [133]. Studies have also shown reduction in overall latency between controller and the managed switches using an SDN based Mobile Core Network (MCN) in comparison with Evolved Packet Core (EPC) [134]. SDN based cellular networks, therefore, offer increased sustainability, lowering operational costs while also allowing operators a global network view for sophisticated traffic monitoring.

Energy efficiency in 5G networking using SDN technology has also resulted in effective energy utilization at multiple levels of 5G functionalities. Within 5G networks, efficient resource management is essential to allow maximum utilization, network slicing, and guaranteeing fairness among several QoS classes [135]. SDN, therefore, has been test-implemented in 5G to allow rapid application service provisioning while adhering to stringent QoS requirements.

Furthermore, furnishing seamless mobility using an almost permanent connection to the mobile network is a fundamental requirement for future cellular services. Software defined Distributed Mobility Management (S-DMM), in this context, overcomes the limitations of legacy mobility management protocols (such as DMM) and is independent of the underlying technologies while offering scalable per flow mobility [136]. Having no significant performance penalty, S-DMM will play a critical role in 5G mobile networks because of ubiquitous connectivity demands and better usage of network resources [137].

With regard to service delivery in mobile cellular networks, information-centric approaches promise enhanced performance, where data may be cached locally at certain points within the 5G network and coordinated by the SDN controller to reduce latency experienced by end users [138, 139]. Applications using the peer-to-peer (P2P) architecture have shown to take advantage of SDN based offloading and redirection of peering traffic, which circumvents the mobile core. Compared to the standard P2P data transfer between smartphone and mobile network, that travels up the cellular hierarchy and is redirected back down to a nearby peering device, intelligent flow forwarding by the SDN controller results in a substantial decrease in overall latency benefiting mobile P2P users [140].

5.1.2. Wireless Mesh Networks. In addition to the employment of fourth- and fifth-generation (4G/5G) mobile cellular technologies, wireless mesh networks (WMNs) have also seen increasing deployment in modern transportation and Internet access systems [141, 142]. While WMNs offer the flexibility of network nodes comprising devices such as routers, laptops, and smartphones to associate and disassociate with a network, dynamic topology, and diversity of device communication requirements make a WMN difficult to manage [143]. Traffic engineering in a network environment

TABLE 7: Summarization of SDN application avenues.

Application Avenue	Brief description	Proposed initiatives
Wireless communications	Using a programmable wireless data plane offering flexible routing and traffic forwarding, SDN has the potential to fine tune mobile communication performance and introduce new applications and services.	<ul style="list-style-type: none"> (i) Introduction of a modular SDN framework in 5G cellular communication realm for horizontal service chaining and resource provisioning as well as analysing contextual analyses of user data to create intelligent traffic forwarding policies. (ii) Simplifying management and traffic engineering in wireless mesh networks and deploying crowd-sharing models to create opportunities for network connectivity and bandwidth sharing. (iii) Optimizing Wi-Fi radio channel assignment and furnishing communication between nodes to extend coverage in view of changing user load. (iv) Increasing scalability and routing autonomy in IoT networking using SDN controllers to reduce management overhead and develop energy efficient control primitives.
Data centers and cloud environments	Operating at large scales such as data centers requires optimal traffic engineering and control which can be facilitated by the SDN framework to reduce latency, improve resource allocation, and reduce operational costs.	<ul style="list-style-type: none"> (i) Reducing network latency, improving control, and introducing intelligent resource provisioning in an automated and dynamic fashion in data centers using SDN based traffic orchestration. (ii) Incorporation of SDN in cloud environments to increase service scalability and automated load balancing in multitenant shares. (iii) Increasing energy efficiency in data center networking using the centralized control plane to place selected devices in low power modes during periods of underutilization.
Campus and high speed networks	Traffic patterns in enterprise campus and high speed backbone networks may show great deal of variation over time, requiring SDN based solutions for real-time programming of the network fabric according to prevailing traffic conditions.	<ul style="list-style-type: none"> (i) Using centralized SDN controller(s) to effectively monitor real-time traffic and accordingly load balance traffic over available links. (ii) Integrating heterogeneous networks using packet based optical communication and circuit switching technologies to allow software defined optical networking (SDON).
Residential environments	Using the SDN framework to allow users and service providers greater visibility into residential and small office network usage for generating subscription models, managing bandwidth, capping data use, and introducing security features.	<ul style="list-style-type: none"> (i) Enhancing monitoring and management of resources using SDN enabled customer routers (data plane) from centralized service provider controller(s). (ii) Implementing anomaly detection systems in programmable residential SDN environments for greater accuracy and scalability of use.

with high device mobility and individual device communication primitives is nonetheless difficult and also prone to security vulnerabilities. Software defined networking due to decoupling of control and data plane provides a suitable framework for the optimization of network resources in WMNs. Recent work has, therefore, seen the utilization of SDN principle for managing wireless mesh networks. The use of OpenFlow based architectures for deployment and management of WMNs has investigated specific areas for improving network management in relation to user mobility, traffic routing, and load balancing [144–146]. Detti et al. [145], for example, proposed the integration of SDN in WMN comprising OpenFlow switches to implement fine-grained traffic optimization. Using the centralized controller, the scheme used Optimized Link State Routing Protocol (OLSR) and OpenFlow protocol to route control and data traffic. The resulting improvements to user performance were significant, when the approach was tested to balance outgoing traffic among multiple Internet gateways of a mesh network using the SDN controller. In terms of mobility management, SDN has been used in projects such as OpenRoads [139] and Open

SDWN [147] to provide seamless wireless network usage for users on the move.

Crowd-shared SDN based wireless networks have also created opportunities to share broadband bandwidth among users, particularly in rural areas with minimal Internet availability [148]. Using network virtualization enabled by SDN can also allow existing rural networks to serve as infrastructure providers for existing ISPs, enabling cooperation while simplifying management [149]. Centralized control of crowd-shared WMNs using SDN may allow more efficient bandwidth sharing in residential environments [143]. While wireless technologies such as Public Access Wi-Fi Service (PAWS) play a significant part in bringing crowd-shared network for benefiting the general public, it also suffers from limited coverage and capacity constraints due to having a single point of Internet access. In comparison, a crowd-shared software defined wireless mesh network (SDWMN) allows greater advantage offering multiple points of Internet access [148]. The crowd-shared network connects the home routers as a mesh. Having advanced knowledge of user sharing policies, the number of flow redirections

from the guest user can be minimized in turn reducing packet reordering and SDN control channel overhead. The SDWMN consequently offers much higher utilization of shared bandwidth as opposed to PAWS based crowd-sharing and is able to accommodate a significantly large volume of guest user traffic.

The incorporation of SDN in WMNs has also facilitated experimentation aimed at benchmarking scalability of WMN solutions as well as understanding trade-offs between in-band and out-of-band communication overhead between wireless nodes and the SDN controller [150–153]. Dimogerontakis et al. [150] proposed an SDN based extension to Community-lab, a wireless community networks test bed, allowing researchers to experiment and manage L2 topology, a feature not available in the original testing platform. Salsano et al. [151] evaluated SDN controller selection in wireless mesh networks to improve control plane resilience and availability. Using SDN controller to manage traffic in WMNs also requires minimizing the control channel overhead between individual nodes and the control plane to optimize wireless spectrum usage, a limited resource. Deployed solutions, therefore, need to consider the pros and cons of having in-band or out-of-band communication for controller to device (wireless node) traffic [152, 153]. The design choices are highly dependent on operational requirements. While out-of-band signalling requires additional infrastructure and could be more costly, in-band control signalling leads to contention between data and control packets sharing the same wireless medium [143]. To alleviate contention in the wireless medium, multiple traffic orchestration algorithms may be used to efficiently divide the available spectrum between control and data traffic.

5.1.3. Wi-Fi Access Networks. The increasing popularity of Wi-Fi networks have led to growing congestion in the nonlicensed spectrum being used for the same. Wi-Fi channel assignment in dense areas remains uncoordinated and somewhat temporal, resulting in negative experience for users frequenting the respective network [154]. Centralized resource management using SDN has therefore motivated work in optimizing radio frequency channel assignment in view of prevailing network conditions. Additionally, novel Wi-Fi deployments using SDN are being employed to support improved network management. Wi-Fi management framework including Odin [155], EmPOWER [156], and OpenSDWN [147] has contributed to efficient SSID selection and device association. Other work has focused on using novel contention management algorithms to optimize access point (AP) selection, identifying the suitability of available wireless resources which meet application requirements, in turn saving bandwidth and increasing user satisfaction [157]. The resulting performance of the proposed architectures significantly improves the Signal-to-Noise Ratio (SNR) for the end users as well as the spectral efficiency at each AP in comparison with conventional approaches [154].

The SDN framework is also being used to furnish cooperation between wireless nodes, extending the coverage to end users who otherwise may not be able to access network service. Specifically in the context of user-centric networking

(UCN) framework, SDN based traffic engineering significantly mitigates the challenges of sharing limited Wi-Fi network capacity efficiently to provide Internet connectivity to end users [158]. Furthermore, user access management challenges in enterprise networks may require resource allocation policies to individual user devices according to predefined usage policies. OpenFlow protocol has been used to offer bandwidth slicing on commercial Wi-Fi APs, implementing dynamic real-time bandwidth adjustments for user devices belonging to different policy groups [159]. The OpenFlow protocol may also be used to control connectivity to a target service using a centralized controller, responsible for cooperatively configuring several wireless base stations and the L2 backhaul network [160]. Centralized resource management reduces delay for time-critical applications such as VoIP, showing conformance level within the IEEE 802.11e standard. Similar to mobility management in cellular networks, SDN could be used to offer a great deal of ubiquity to users connecting to different wireless infrastructures belonging to multiple providers through user device identity management which is coordinated and proactively managed by the controller [155].

5.1.4. Internet of Things. The Internet of Things (IoT) paradigm comprises a diverse set of devices and protocols requiring a great deal of integration and effective network resource management among the underlying heterogeneous components. Leveraging SDN along with NFV appears to be a viable choice to introduce scalability in device management and optimize the routing of data generated from the respective IoT sensors [161]. Several prototype implementations have been considered for this purpose ranging from the use of programmable WSNs which forwards packets according to rules installed by the SDN controller [162, 163], to giving more routing autonomy to individual sensors incorporating low-end controller functionality [164, 165]. The pros and cons of each scheme again depend on the deployment scenario and the higher level IoT-specific application requirements. While low-level controller functionality in IoT devices may serve to reduce control channel overhead and the latency involved in installation of traffic flows, computational capability and power constraints may make such a scheme impractical. Specific service requirements for IoT networks can be translated by the controller into network requirements (minimum data rate, delay, packet loss, etc.) to furnish optimized end-to-end performance [166]. The centralized SDN control plane may also be used to dynamically offer differentiated quality levels for different types of IoT tasks.

The wireless mesh networks (WMN) detailed earlier provide a common baseline topology for IoT device communication. As such it may be argued that OpenFlow enabled WMNs may serve as catalyst for incorporating SDN principles in IoT environments. IoT devices may connect to OpenFlow based mesh routers using the 802.11 interface, while the central controller facilitates the installation of traffic forwarding in routers and allocates channel assignments to avoid wireless contention [167]. Further to traffic routing, to optimize energy efficiency the SDN controller can also be used for selecting sensor nodes (based on remaining power) to be put

in sleep mode based on calculations of neighbourhood nodes which are awake and can continue to provide connectivity [168]. The subsequent reduction in node broadcasts that are inherent in a de-centralized control environment improves the energy efficiency and lifetime of the IoT network. The inclusion of SDN in IoT remains an open application avenue requiring bespoke IoT controller and network architectures to adequately address the intrinsic heterogeneity prevalent in the IoT ecosystem.

5.2. Data Centers and Cloud Environments. Optimal traffic engineering, network control, and policy implementation are an absolute requirement when operating at large scales in data center environments. Increased latency, faults, and prolonged troubleshooting may result in not only negative end user experience but also significant cost penalties for operators. Data center (DC) SDN implementations, therefore, through a centralized control framework monitor and manage hundreds of network devices and services promising effective resource provisioning for operators. Google, for example, has used SDN technology to connect its geographically dispersed data centers around the globe allowing increased resilience and manageability [169].

Cloud computing has also seen the integration of SDN based traffic engineering solutions to increase service scalability and automated network provisioning. A prominent example is the Microsoft's public cloud described in [170]. The study highlights SDN based load balancing solution Ananta, a layer 4 load balancer employing commodity hardware to provide multitenant cloud management. Using host agents, packet modification is localized enabling high scalability across the DC. The project has seen significant deployment in the Microsoft Azure public cloud allowing high throughput for several tenants allocated a single public IP address. Another prominent SDN deployment in cloud environment is NTT's software defined edge gateway automation system [171]. The gateway uses OpenFlow protocol for maximum flexibility in network provisioning and evaluates possible extension to existing OpenFlow features baselining the SDN stack to allow robust cloud gateway deployment.

On a separate strand reducing energy consumption in data centers has also been an area of focus for operators to reduce operational costs. Since most DCs are overprovisioned for peak traffic the energy efficiency during periods of underutilization is minimal. SDN technologies such as ElasticTree allow network-wide power management by switching off redundant switches from the controller side during low traffic demand [172]. SDN implementation in DCs till present remains one of the primary beneficiaries of the framework.

5.3. Campus and High-Speed Networks. Enterprise networks may show a great deal of variability in traffic patterns requiring proactive management to adjust network policies and fine tuning performance using a programmable SDN framework. A centralized control plane may also aid in effective monitoring and utilization of network resources for readjustment. An additional benefit may be to eliminate middle boxes providing services such as NAT, firewalls, access control, service differentiation solutions, and load balancers

[173–175]. Inclusion of an SDN controller for optimal network provisioning while offering simplicity also allows external third-party network administration of the enterprise network and increased support for virtualization.

The integration of heterogeneous networking technologies using OpenFlow enabled network elements and a centralized controller has seen a great deal of applicability in optical networking. Using centralized real-time programmability, SDN enabled hardware from multiple vendors and optical packet based as well as circuit-switched networks can be placed under the SDN controller. Gudla et al. [176], for example, used NetFPGA [22] along with Wavelength Selective Switching (WSS) facilitated by OpenFlow protocol. High-speed optical communications require an appraisal of the existing OpenFlow framework and possible extensions to achieve a higher level of integration [177]. Liu et al. [178] used virtual Ethernet based interfaces to demonstrate OpenFlow based wavelength path controlling in optical networking. A commodity SDN controller, such as NOX and POX, can operate the optical light paths using OpenFlow by mapping virtual Ethernet interfaces to physical ports of an optical cross-connect node. The evaluation of network performance metrics included latency of path setup and verification of routing and wavelength assignment allocation using dynamic node control provided promising results for future software defined optical networking (SODN).

In comparison with typical distributed GMPLS protocol, SDON using a unified control protocol for QoS metrics offers greater capacity and performance optimization in optical burst switching [179]. With increasing use of fibre technologies in enterprise networks, the Optical Transport Working Group (OTWG) created by the Open Network Foundation (ONF) envisions applying southbound protocols such as OpenFlow to improve optical network management flexibility [180]. The application of SDN and in particular OpenFlow based controls in high-speed and campus networking, therefore, continues to grow resulting in new as well as hybrid solutions to achieve greater network programmability.

5.4. Residential Networks. Software defined networking has also been considered as an efficient means to manage residential and small office networks. One of the fundamental requirements of such networks is operators and residential users having a greater degree of visibility into usage through effective monitoring using the SDN paradigm [181–183]. To relieve the burden of network management on residential gateways, Dillon and Winters [184] discussed creation of virtual residential gateway (data plane) using software defined networking controller at the service provider side to remotely allow management flexibility innovative service delivery in homes. The residential router or gateway may be controlled and managed remotely via an SDN controller at the service provider premises, with the latter mainly responsible for fine tuning and troubleshooting the residential network [182, 184, 185].

Management of residential networks presents a key challenge for residential users and service providers alike, the benchmarking of home user activities through collection of

traffic metrics, and the setup involved. While some proposals discuss the implementation of custom logging platforms to collect such usage information, it is also noted that the framework presents privacy and scalability issues if external service provider is allowed with such a detailed insight into user activities [183]. Therefore, giving residential users rather than service providers more control while using SDN based monitoring in the residential environment to tune network policies also needs to be considered. Using localized application monitoring mechanism [185, 186], users in each residential premise may acquire an enhanced view of their application trends. A policy primitive using application priorities defined by the end user can afterwards be employed to distribute last mile network bandwidth between the residential router and service provider gateway. In addition to simple application prioritization SDN resource allocation framework may also be used to distribute the last mile bandwidth for users in proportion to their resource consumption. Incorporation of the SDN framework in residential networks offers improved scalability and privacy for network management [186].

From a security perspective it has been argued that an anomaly detection system in a programmable residential SDN provides more accuracy and higher scalability than intrusion detection systems deployed at Internet service provider side [187]. Feamster in [182] proposed completely outsourcing residential network security harnessing programmable network switches offering flexible remote management. Employing the outsourced technical expertise management and running of tasks such as software updates and antivirus utilities may become more effective as the external operator has a wider view of network activity, although the privacy of end users as mentioned in [183] is equally important consideration in a more realistic residential network management framework where technical operations are outsourced. A review on primary security challenges in SDN is detailed in Section 6.4. The inclusion of SDN framework in residential networking therefore remains an active area study in academic and industry use-cases.

6. Research Challenges

Increasing application of SDN framework in several network settings have also highlighted areas of concern ranging from application performance to security inadequacies inherent in the present architecture. The following subsections highlight the primary investigations and research advances made in the SDN domain. A summarization of the areas of concentration and subsequent research initiatives is presented in Table 8.

6.1. Application and Service Performance. Traffic optimization carried out on the basis of network application type to a significant extent focuses on increasing specific service performance in SDN. The concentration in this domain has particularly seen various studies concentrating on video and voice streaming, using the SDN architecture to classify the respective applications and dynamically define flow forwarding policies to improve quality of service. Proponents of application-aware SDN infrastructure consider the

benefits the framework brings to offer enhanced performance isolation for specific applications. While southbound APIs such as OpenFlow are capable of Layer-2/3/4 based policy enforcement they lack the high level application awareness. Qazi et al. [188] in an application-aware SDN study discuss Atlas, a crowd sourcing approach deploying software agent on user devices to collect the netstat logs sent to the control plane. Using machine learning techniques the scheme identifies approximately forty applications from Google Play Store. The study then uses SDN framework applying predefined policy actions to the respective flows as well as collect flow statistics. Similarly Mekky et al. [189] propose per application flow metering using the SDN framework. Applications are identified in the data plane and the relevant policies applied using application tables, limiting the control channel overhead towards the SDN controller. When tested and implemented using a content-aware server selection application and multiple virtual IP pool of services the study showed significantly good application forwarding performance with low overhead.

Concentrating on video streaming applications Egilmez and Tekalp [190], devise an analytical framework for traffic optimization at the control layer enabling dynamic quality of service (QoS). The study reported significant improvement for streaming of encoded videos under several coding configurations and congestion scenarios. Jarschel et al. [191], focused on YouTube streaming in particular. The study used Deep Packet Inspection (DPI) and based on information input showed how application detection and the different application metrics can be used to enhance the Quality of Experience (QoE). Ruckert et al. [192] developed Rent a Super Peer (RASP), an architecture offering streaming P2P videos on the OpenFlow network. Another example of video optimization is the CastFlow [193] which proposes a prototype for IPTV, emulated and tested over Mininet utility. Noghani and Sunay [194], consider a multiple description coded streaming video multicast framework using SDN framework to realize significant performance gains. Other studies seek to substantiate the importance of underlying test beds on the evaluated improvements to video quality. Panware et al. in [195], for example, benchmark the packet delay and latency performance of videos tested on Mininet as well as actual physical PC clusters using Open vSwitch. The Microsoft Lync platform [196] also offers a prominent test case example of an application using SDN based network abstraction to optimize real-time messaging, video and voice communication among Lync clients. Microsoft released a purpose built Lync northbound API for SDN that gives administrators visibility in to voice, video and media stream metrics deployed in enterprise environments. Lync SDN API, as per Microsoft can immediately enhance the diagnostic capability of monitoring Lync communication in SDN as well improve QoS. The effects of Lync and other similar targeted specific service improvement on other applications in the enterprise network and the resulting overall experience of end users remains to be considered.

Recent advances in virtualization technologies have seen a range of applications being hosted on multiple servers in cloud environments and private data centers. SDN again due

TABLE 8: Summarization of SDN research initiatives.

Area of concentration	Brief description	Research initiatives
Application performance	Improving the performance of individual network applications and services in the SDN framework using novel optimization techniques in wired, wireless, and heterogeneous settings.	(i) Increasing SDN <i>application awareness</i> and optimizing time-critical application services using flow metering. (ii) Development of SDN <i>monitoring tools</i> for evaluating performance gains in heterogeneous network environments. (iii) <i>Embedding network services</i> , such as authentication, firewalls, proxies, and so forth, in the data plane fabric. (iv) <i>Information-centric</i> approaches exploiting location-based data caching.
Controller scalability and placement	Controller placement in large SDN environments offers a complexity optimization problem affecting latency, capacity, and fault tolerance. The design of the control plane remains a multifaceted topic of several research studies.	(i) <i>Reducing latency</i> by solving optimal controller placement problem (NP-hard). (ii) Solutions <i>minimizing controller workload</i> with respect to controller placement. (iii) <i>Distributed control architectures</i> offering <i>greater reliability</i> using heuristic and greedy algorithms to refactor larger network into smaller subnetworks. (iv) <i>Combinatorial approaches</i> optimizing multiple network performance metrics in relation to controller placement, providing a trade-off between performance gains and operational requirements.
Switch and controller design	Studies aimed at improving northbound API standardization among multiple application platforms, level of control delegation appropriate for data plane elements, and optimal hardware architectures.	(i) <i>Standardization of the northbound API</i> , involving studies in designing a vendor neutral policy abstraction language offering vertical and horizontal integration with existing network fabric/services. (ii) Greater level of <i>control delegation</i> to network switches aimed at reducing controller overhead and increasing fail-safe redundancy. (iii) <i>New architectures</i> for controller and switch design.
Security	SDN due to centralized network control creates potential security challenges directed at control plane (traffic) and data plane elements including network appliances and middle boxes.	(i) Designing SDN security reference models and methods focusing on securing network entities to avoid disruption and security compromises. (ii) Control channel and application-controller traffic monitoring and anomaly detection in specific avenues for example, clouds. (iii) Network/state information storage and retrieval for postevent and forensic examination.
Interconnecting SDN Domains	Autonomous systems managed by independent SDN controllers (or operators) requiring timely intercontroller information exchange and underlying mechanisms to support network orchestration as well as application delivery between disparate SDN domains.	(i) <i>Legacy routing protocols</i> such as BGP and OSPF implemented <i>with extensions</i> to support interdomain SDN controller communication. (ii) <i>SDN-specific approaches</i> defining new network orchestration applications and information exchange architectures.

to decoupling of control logic from forwarding elements is seen as a key enabling technology in this domain. Monitoring and improving the performance of hosted applications offers development of niche tools able to monitor the application traffic in virtual platforms and apply traffic management policies. Liu and Wood [197] describe NetAlytics, a platform for large scale performance analysis which uses NFV technology to deploy software-based packet monitors in the network and an SDN management overlay to direct packets flows to these monitors. The system aims to diagnose application performance issues and the collected statistics offer administrators insight into application popularity. Maan et al. [198] developed a system for monitoring network flows at the edge, closer to the users in cloud based data centers. The work explores enabling flow monitoring in virtual switches in servers. EMC2 is proposed, a scalable

network monitoring utility in cloud data centers to be used in tandem with performance evaluation of various switch flow monitoring techniques. The evaluation recommends NetFlow [199], giving a very good network coverage with minimal use of resources to monitor application traffic in virtual environments and cloud based data centers. Hwang et al. [200] discuss NetVM, a framework allowing customizable data plane processing services including firewalls, proxies and routers to be embedded with virtual servers using SDN. The authors highlight the benefits achieved in dynamically scaling, deploying, and reprogramming of the embedded network applications. Individual application performance has been the typical approach to improve end user experience in both legacy and SDN environment, isolated service improvement does not specifically consider the mix of user application trends in a typical network environment while

defining network policy primitives. The end users may have diverse application trends, and traffic forwarding constructs optimizing a single (or subset) of applications in the SDN framework might result in performance penalties for users frequenting a different (subset) of applications. The profiling of user traffic may therefore enable the extraction of user traffic trends and utilized as a means to aid the SDN controller in balancing network resource share among several types of users (profiles) in residential networks.

Another category of work in application performance improvement proposes an information-centric approach to optimized service delivery in SDN. The motivation behind the studies is the fact that while networking and present Internet usually exploits location-based addressing and host-to-host communications. Addressing data by name (Named Data Networking) and distribution over dispersed locations may offer optimization for application content delivery to end users. Information-centric content delivery network may then be combined with SDN to allow greater deal of network programmability and a key enabler for content distribution. Studies including [201–206] propose the use of SDN and OpenFlow to support customized matching of packet headers for service delivery to end users from content servers.

Application delivery and service improvement to a greater extent also depends on the architecture and suitability of network elements in the data plane to implement customized traffic forwarding policies efficiently. The present and future trends in networking highlight the fact that heterogeneous networks ranging from wired, wireless, cellular, ad hoc, to vehicular environments and their inter-connection capability, together exponential growth in data usage, [207] will play a crucial part in application delivery. SDN, therefore, may offer operators the ability to integrate and share capacity on different shared physical media, a substantially challenging task with legacy networking infrastructure [208]. Application services in networks with heterogeneous characteristics such as topology, physical medium, and stability of connections can potentially use the SDN paradigm for routing and resource allocation. A few studies [139, 144, 209] have, therefore, examined the scope of the application delivery using SDN in infrastructures including WiMAX and Wi-Fi access. The OpenRoads [139] project, for example, discusses seamless user service delivery between multiple wireless infrastructures. Other works [144, 206] offer enhanced application performance in wireless mesh environments using OpenFlow.

While the above studies promise and further the network performance of individual applications, they do not, however, specifically consider the consequences of such isolated performance boosting on other application traffic traversing the SDN fabric. As highlighted earlier, the prevalent work in traffic optimization in SDN focuses on improving the quality of individual applications and services such as video streaming or voice communications. Other studies involving information-centric networking focus on bringing the data sources closer to the network edge, to again improve traffic conditions for the hosted application(s). Regardless of whether the SDN is comprises of heterogeneous or monolithic networking fabric, in realistic environments,

users may frequent a range of applications albeit in different proportions. Optimization efforts, therefore, may need to consider the mix of applications being used and the performance caveats for end users as a result of individual service improvement.

6.2. Controller Scalability and Placement. The SDN controller manages switches providing traffic forwarding rules which affect the packet behaviour. In larger network environments, however, the placement of the controller is pertinent to optimize network connectivity [210]. The connections between controller(s) and switches in the data plane require low latency for seamless operation [211]. To address scalability, it is, therefore, usually required having more than one controller serving the data plane. Multiple controllers may also be used to offer a greater level of redundancy in critical network infrastructures. Prevalent research in this area aims to deploy the multiple controllers in several network locations and as such determining the exact point of placement becomes critical [210–214].

The placement problem was first identified by Heller et al. in [211] as an NP-hard problem focusing on discussing the determination of exact number and optimal location for SDN controllers. The study highlighted that the best controller placement solution should minimize the controller to switch control traffic latency. Similarly, Sallahi and St-Hilaire [212] considered the placement problem from an operational perspective discussing the cost involved in deploying and installing controllers and connecting these in the network fabric. The studies used traversal search perusing through the best solutions to find the optimal candidate, a time consuming process proportional to the size of the network. Some of the topologies took an order of magnitude greater than 30 hours to find a satisfactory placement solution.

Yao et al. [213] focused on the controller workload and that placement should also consider that the load is not exceeding the controller capacity. The proposed placement solution used k -center algorithm [215] to minimize the value of k (controllers) that meet the capacity requirements. In later work, Yao et al. [216] discussed placing controllers at network hotspots where switches carry most of the traffic. Switches may utilize less overloaded controllers migrating from one to another with changing traffic demand.

Ros and Ruiz [217] considered network reliability highlighting positive correlation between fault tolerance and controller placement. The study used heuristic algorithms to compute the placement to meet network reliability determining the maximum number of controllers which may be deployed. Zhang et al. [210] used min-cut method discussed in [218] to compute the maximum number of disjoint paths which separate the network into smaller networks, each having its own controller. Similarly Guo and Bhattacharya [219] generated a hierarchical tree [220] of network nodes, dividing it into k clusters or subnetworks. Nodes with maximum closeness to other nodes were selected for controller deployment. In [221] a greedy algorithm was used to enhance controller placement reliability in the event of network state changes as well as single link failures in tandem with the optimal controller placement. Hu et al. [222] used multiple

algorithms including 1-w greedy, simulated annealing and brute force with brute force offering the best optimal solution, ignoring the time consumption involved.

Other notable works such as Onix [63], HyperFlow [64], and Kandoo [65] proposed a distributed control architecture allowing some level of scalability and reliability in large SDNs. The schemes allow for multiple controllers to manage the data plane. HyperFlow details a flat or horizontal scaling of controllers, while Kandoo and Onix use a hierarchical structure. The distributed architecture is comprised of root and localized controllers, each level having a different view of the underlying network. Framework such as Difane [223] and DevoFlow [224] distributes some control functions to the SDN switches to reduce the controller overhead. Offloading partial workload helps in improving network scalability, however, requires significant modifications in switching hardware to accommodate functional requirements. SDN controller placement, therefore, remains an active area of research with several solutions and approaches pursued to achieve a greater deal of scalability allowing greater network performance.

6.3. Switch and Controller Design. Innovations and proposals in controller and switch design seek to circumvent some of the problems associated with policy implementation and simultaneously address additional areas needing improvement such as controller and switch scalability. Controller-switch interaction is served by standard southbound APIs such as OpenFlow [23], XMPP [26], or ForCES [14]; however, a similar level of standardization is not available at application-controller northbound interface. Since the northbound interface purely relies on the application logic, proponents of nonstandardization of northbound API argue that the present framework allows for greater degree of innovation with custom northbound communication fitting the application or service using the SDN. As previously mentioned, there are a number of controller utilities and platforms available using which application can interact with each other as well as the underlying network elements for traffic engineering. The application developer, however, needs an in-depth knowledge of the controller implementation to deploy application APIs.

A few proposals have highlighted the need for network configuration language that can express the administrator policies seamlessly on the underlying controller implementation [225–228]. Policy description language such as Procera [225] and Frenetic [226] builds a policy layer on existing controllers, interfacing with configuration, graphical interfaces, and other network monitors to translate the administrator defined policies to flow level details which can be used by the controller. Other examples exploring network configuration languages include FML [227] and Nettle [228]. In [225], it is proposed that network configuration and management focusing on changing network conditions and state using policy definitions. The northbound communication API may further allow the SDN to apply segregated policies on same application flows based on destination or source IP address. Monsanto et al. [229] on the other hand use modularized policy implementation which ensures

that flow rules for one network application task do not interfere or replace rules for other tasks. As mentioned earlier, the chances of ONF or the industry standardizing the northbound API look slim and operators will continue to develop and deploy custom northbound APIs, until there is a clear and persistent need for a universal northbound interface.

In terms of switch design innovation, research works resulting in schemes such as Difane [223] and DevoFlow [224] aim to reduce the number of requests being sent from switches to the controller and in hindsight bring a relatively greater degree of control delegation to the data plane. Proactive implementation of policies in the switches leads to a substantial lowering of control overhead generated during real-time operation. Luo et al. [230] discuss the replacement of ASIC based counters for rule-matching in switches to ones processed in the CPU. Other technologies such as FLARE [231] allow for complete programmability of not only the data and control planes but also the control channel between them. A single controller may be able to handle up to 6 million to 12 million flows per second [46, 232]. However, lowering propagation latency and increasing fault tolerance and robustness have seen development of controller architectures using horizontal and hierarchical clusters [63–65].

Although network control and application traffic management have been the focus of much of the previous work, new architectures for controllers and switches also remain an active area of academic and industry research.

6.4. Security. Increasing interest in SDN in the networking community has also initiated significant debate highlighting the security challenges inherent in an SDN framework. The OpenFlow switch specification [23] includes relatively basic security incorporation in SDNs using optional transport layer security (TLS) allowing mutual controller-switch authentication without specifying the exact TLS standard. TLS, although, has not been given much thought in several open source controller and switch platforms which may lead to anomalous rule insertion in flow tables [233].

Centralized control makes the scheme vulnerable to attacks directed at the control plane which may disrupt the entire network. The intelligence in the centralized control plane may offer hackers the opportunity to explore security vulnerabilities in the controller and take over the entire network [234, 235]. On the flip side, it is argued that the information generated from traffic analysis or anomaly detection in the network can be regularly transferred to the SDN controller, having a global network-wide view to analyse and correlate feedback for efficient security [236].

In addition to securing the controller, targeted attacks on the network (e.g., DDoS) and subsequent controller failure or exhaustion of limited flow tables in network switches may result in substantial network service downtime until the controller is up and running and the threat has been mitigated. The control channel between the controller and network devices also has to be secure enough to reject anomalous injection; the same is true for application-controller communication. Effectively establishing trust among all the

network devices and the applications on top are, therefore, considered a key security concern. Vulnerability analysis, mitigation studies and a standardized framework for SDN security has, therefore, been the focus of multiple deliberations [237–240] with a great deal of focus in the security domain laid on controller-switch and intercontroller communication. Shin and Gu [237], for example, undertake vulnerability evaluation of SDN by evaluating the feasibility of finger-printing attacks. The evaluation finger-prints the SDN gear such as OpenFlow switches in the networks and targets the respective elements with denial of service (DoS) attack on the controller via control channel and the data plane elements by exploiting flow tables. Both entities are identified as significant areas of SDN vulnerability. Similarly, Smeliansky [238] discussed communication protocol security with consideration for infrastructure and software services, concluding that control-data plane and control to control plane communication requires substantial hardening to mitigate security threats. Some of the solutions in communication challenges point to replication of SDN controllers and network applications to provide redundancy and fail-safe operations arising due to misconfigurations and software bugs [241]. Other investigations propose entity or service mobility (moving defense), to counter security threats [233]. The controller functionality, for example, could be continuously shifted across several network elements making targeted attacks on the controller more challenging for those seeking to exploit the control plane.

SDN security is also considered in several application contexts, for the technology to gain wider acceptance in particular avenues including wireless communications and cloud computing [239, 240, 242]. Schehlmann et al. in [239] discuss potential improvements in network management costs, as well as attack detection and mitigation by using SDN framework itself as a potential barrier to security vulnerabilities. SDN enables the incorporation of certain security functionalities through decoupling of network control from forwarding logic where traffic filtering can be achieved using key traffic (packet) identifiers usually requiring dedicated firewalls and intrusion detection/preventions systems in legacy networking. Additional security layers may be added on top of existing SDN layers as well as introduction of agents in data plane elements to introduce more granularities for filtering traffic specific to heterogeneous networks such as wireless settings as investigated in [240]. Similarly, in SDN enabled cloud computing, additional security may be introduced at each SDN layer based on underlying operational requirements to make intra- and intercloud communication more secure [242].

In addition to specific application avenues, generally real-time SDN monitoring has to be robust enough to offer timely detection of anomalous network events and containment [187]. The monitoring information not only provides insight into traffic but similar to legacy networking, may also be the focus of storage to satisfy technical as well as legal requirements. Some work in this area, details the consideration which should be given to monitoring storage, for subsequent perusal in conducting and aiding forensic analysis [175].

Security approaches may focus on securing the network itself by embedding intelligent security alarms in the network elements such as switches and controllers or include SDN oriented security utilities in the functional entities such as application servers and storage clusters. Organization, such as OpenFlowSec [243], particularly focuses on security challenges presented by the SDN paradigm and OpenFlow enabled devices. Development work has considered designing reference implementations of security features at different layers of the OpenFlow stack. A detailed taxonomy of security threats in the SDN paradigm is presented in [244]. Beyond the basic SDN architecture, the deployment of robust security in the SDN paradigm is still very much an area requiring further study. It is, however, a widely held belief that without a significant increase in focus on SDN security, the paradigm may not see adoption beyond private DC infrastructures or autonomous organizational deployments.

6.5. Interconnecting SDN Domains. Autonomous SDN environments spanning over several data center networks geographically dispersed customer sites as well as mobile entities require network traffic orchestration across interdomain/intradomain subnets of SDN controllers. Seamless operation of applications across SDNs managed by independent controllers (or operators) requires timely information exchange and the underlying mechanisms to support communication between disparate SDN domains. While horizontal (east-west) and vertical control plane distribution schemes highlighted earlier in Section 6.2 offer scalability in SDN controller placement, intercontroller communication specifically requires information exchange protocols to carry network related information among different SDN systems. In this context, legacy protocols such as BGP [245] and OSPF [246] have been used to interconnect SDN domains and establish communication between respective SDN controllers. Existing BGP message format can be customized allowing communication of QoS policies and other parameters including network topology, traffic conditions, security events and network failures across independent SDN domains [247]. The OFBGP [248] offers, such an SDN based solution, dividing information exchange functionalities into two modules, the *BGP protocol* responsible for connection maintenance with the BGP neighbours and *BGP decision* for computing respective routes. SDN applications can leverage the resulting parallelization in computing routing decisions and the OFBGP framework easily scales out with increase in the number of BGP (controller) neighbours. Having a built-in fault remedial process, OFBGP also allows restoring SDN to a previously working (backup) state. BTSDN discussed by Lin et al. [249] offers a framework aimed at retaining current BGP and legacy BGP border routers to interconnect OpenFlow based SDNs to the rest of the Internet. Chen et al. [250] detailed a mathematical model for reducing BGP convergence time in legacy networks using SDN. The proposal discusses utilizing a greedy algorithm to select existing autonomous systems for incremental SDN deployment, providing improved convergence time when tested on realistic Internet topologies. You et al. [251] focused on designing

a novel interdomain multipath flow transfer mechanism using SDN framework alongside BGP. A source domain (SDN) controller analyses BGP notifications to determine path diversity among different domains and uses interdomain network state exchanges to improve application throughput across networks. Caria et al. [252] implement hybrid SDN operation across multiple autonomous systems using Open Shortest Path First (OSPF) protocol. The scheme partitions an OSPF domain into subdomains using SDN enabled switches as border nodes while employing the SDN controller to tune routing protocol updates among the subdomains. Hybrid deployments again serve as incremental steps in updating legacy network settings to SDN.

In addition to using legacy routing protocols, a number of studies have also focused on designing SDN-specific interdomain routing mechanisms. A Multidimensional Link Vector (MLV) network view exchange mechanism was proposed in [253], offering interdomain routing control by exchanging multiple fields in the IP header, a mechanism which can be deployed as an application over SDN controllers. Wang et al. [254] devised a new routing control plane (RCS) enabling network function chaining across interdomain SDN paths by abstracting and disseminating network functions between autonomous SDNs. The proposal allows customers to set up per application routing paths and can also be applied incrementally to legacy interdomain networking for achieving greater application path diversity. From a network architecture standpoint, distributed architectures, such as DISCO (Distributed SDN Control Plane), allow individual controllers to manage their respective domains and communicate with each other to provide end-to-end network services in an open and extensible manner [255]. The communication primitive uses a lightweight control channel among controllers to share aggregate network-wide information relating to interdomain topology updates, application priority, and virtual machine states. Thai and de Oliveira [256] propose the Interdomain Management Layer (IML), a framework utilizing horizontal network slicing and allowing SDN networks to become autonomous peers with other autonomous systems, while maintaining centralized, operator-defined policy control and facilitating end-to-end flow control across interdomain SDN boundaries. Furthermore, projects such as RouteFlow [69] provide legacy IP routing services over OpenFlow hardware using a virtual network resource pool that depicts the physical topology and utilizes legacy routing engines (e.g., Quagga) [257] to implement end-to-end resource allocation policies. In addition to inter-SDN communication studies relying solely on existing and new protocol extensions, another approach discusses the use of Software Defined Internet Exchange Points (SDX) to convey routing and network related information to participating SDNs [258]. The SDX applies a representative set of combined policies for participants (autonomous networks) based on their full routing table advertisements, which results in efficient conflict resolution and minimum convergence time in response to any future configuration changes and network updates in the respective networks.

A similar industry effort, the Cardigan project [259] by Google, provides an Internet exchange that participants can

connect to and advertise their networks. SDN domain can be connected to the Internet exchange allowing route advertisement into the domain and offering routing of application traffic across it and on to the exchange. The OpenDaylight project enables interdomain SDN controller communication through a Software Defined Network Interface (SDNi) application for the ODL controller [260]. Employing the existing ODL-BGP plugin, the SDNi wrapper exchanges information data (enhanced NLRI updates) through REST APIs as well as stores this information in an SQL database. The application allows each controller to have real-time (controller) peer data during a session while also offering restriction of information exchange(s) based on security policies. Other projects, such as IETF I2RS [28], aim to allow applications greater control in modifying routing decisions using traditional distributed routing protocols executed on network devices. A few industrial case studies have also employed the Session Initiation Protocol (SIP), typically used to establish multimedia sessions, to offer load balancing and peering across SDNs [261, 262]. However, as discussed in [262], the scope of SIP to exchange routing information in inter-SDN domains is rather limited compared to protocols such as BGP, owing the customization allowed by the latter. Inter- and intradomain SDN communication remains an active field of investigation with the ONF proposing the use of existing protocols or extension of these as required to enable routing and network state exchanges in the SDN architecture [1, 5].

7. Conclusion

The present paper highlighted the state of the art in software defined networking (SDN) technologies in detail. Key components of the framework, the northbound and southbound control plane communication interfaces, allow several prominent new and legacy protocols to be used in the SDN paradigm. The influence of few APIs including OpenFlow used for switch-controller communication and REST and Java based OSGi on the application-controller interface has although been profound, resulting in wider adoption of the respective technologies in academic and industrial deployments. In addition to communication APIs, the past few years have seen the development of several SDN controller platforms as well as simulation and debugging technologies, introducing substantial variety in network programmability for academia and industry to experiment and explore. However, despite promising simplified network management and centralized control, research and industrial test cases have also highlighted SDN limitations which require significant improvement. SDN deployment in avenues such as data centers, cloud computing, and wireless communications presents unique operational challenges ranging from application performance and increasing SDN controller scalability to optimizing switch design and securing controller-switch control channel. To further increase SDN acceptance and implementation in realistic networking environments, industrial efforts and research studies, therefore, continue to explore solutions to these challenges allowing greater

realization of SDN technology in existing and future IT infrastructures.

Competing Interests

The author declares that there are no competing interests regarding the publication of this paper.

References

- [1] SDN Architecture, Issue 1, Open Networking Foundation, 2014, https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH.1.0.060620-14.pdf.
- [2] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Computer Networks*, vol. 71, pp. 1–30, 2014.
- [3] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.
- [4] N. McKeown, T. Anderson, H. Balakrishnan et al., "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
- [5] Open Networking Foundation (ONF), <https://www.opennetworking.org/>.
- [6] Open Networking Research Center (ONRC), <http://onrc.stanford.edu/>.
- [7] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2015.
- [8] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.
- [9] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: a comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.
- [10] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: from concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.
- [11] K. Greene, "TR10: Software-defined networking. MIT Technology Review," <http://www2.technologyreview.com/article/41-2194/tr10-software-defined-networking/>.
- [12] J. P. Ronayne, *The Digital Network Introduction to Digital Communications Switching*, Howard W. Sams & Co., Inc., Indianapolis, Ind, USA, 1st edition, 1986.
- [13] E. S. Raymond, *The Art of Unix Programming: Origins and History of Unix, 1969–1995*, Online Book, 2013, <http://www.catb.org/esr/writings/taoup/html/index.html>.
- [14] ForCES, <http://datatracker.ietf.org/doc/rfc3746>.
- [15] L. De Ghein, *MPLS Fundamentals*, 2006.
- [16] PCE, <http://datatracker.ietf.org/wg/pce/>.
- [17] A. T. Campbell, I. Katzela, K. Miki, and J. Vicente, "Open signaling for ATM, internet and mobile networks (OPENSIG'98)," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 1, pp. 97–108, 1999.
- [18] A. Doria, F. Hellstrand, K. Sundell, and T. Worster, "General Switch Management Protocol (GSMP) V3," RFC 3292 (Proposed Standard), June 2002.
- [19] A. Greenberg, G. Hjalmtysson, D. A. Maltz et al., "A clean slate 4D approach to network control and management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, 2005.
- [20] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 1–12, 2007.
- [21] OpenWrt, <https://openwrt.org/>.
- [22] Netfpga platform, <http://netfpga.org>.
- [23] OpenFlow Specification (ONF), <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>.
- [24] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," in *Proceedings of the DARPA Active Networks Conference and Exposition (DANCE '02)*, pp. 2–15, Washington, DC, USA, May 2002.
- [25] Devolved Control of ATM Networks, <http://www.cl.cam.ac.uk/research/srg/netos/old-projects/dcan/#pub>.
- [26] P. Saint-Andre, *XMPP The Definite Guide*, Safari Book, O'Reilly, 2009.
- [27] Y. Lee, G. Bernstein, D. Dhody, and T. Choi, "ALTO extension for collecting data center resource in real-time," ALTO, <https://datatracker.ietf.org/doc/draft-lee-alto-ext-dc-resource/>.
- [28] I2RS, <https://datatracker.ietf.org/wg/i2rs charter/>.
- [29] Cisco OnePK, <https://developer.cisco.com/site/onepk/>.
- [30] R. Enns, "NETCONF Configuration Protocol," RFC 4741 (Proposed Standard), Obsoleted by RFC 6241, December 2006.
- [31] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin, Simple network management protocol (snmp), rfc1157, 1990.
- [32] CERIAS: GeoPlex: Universal Service Platform for IP Network-based Services—10/17/1997, 2014, <http://www.cerias.purdue.edu/>.
- [33] J. E. Van der Merwe, S. Rooney, I. Leslie, and S. Crosby, "The tempest—a practical framework for network programmability," *IEEE Network*, vol. 12, no. 3, pp. 20–28, 1998.
- [34] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: realistic and controlled network experimentation," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 3–14, 2006.
- [35] N. Feamster, L. Gao, and J. Rexford, "How to lease the internet in your spare time," *SIGCOMM Computer Communication Review*, vol. 37, no. 1, pp. 61–64, 2007.
- [36] M. Mahalingam, D. Dutt, K. Duda et al., *VXLAN, A Framework for Overlaying Virtualized Layer 2 Networks Over Layer 3 Networks*, Internet Engineering Task Force, 2011.
- [37] M. Sridharan, K. Duda, I. Ganga, A. Greenberg, G. Lin, M. Pearson et al., NVGRE: network virtualization using generic routing encapsulation, Internet Engineering Task Force, September 2011.
- [38] B. Davie and J. Gross, *STT: A Stateless Transport Tunneling Protocol for Network Virtualization (STT)*, Internet Engineering Task Force, 2012.
- [39] Cisco ACI, <http://www.cisco.com/c/en/us/solutions/data-center-virtualization/application-centric-infrastructure/index.html>.

- [40] Q. Duan, Y. H. Yan, and A. V. Vasilakos, "A survey on service-oriented network virtualization toward convergence of networking and cloud computing," *IEEE Transactions on Network and Service Management*, vol. 9, no. 4, pp. 373–392, 2012.
- [41] Cisco OpFlex, <http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-cl1-731302.html>.
- [42] R. T. Fielding, "Chapter 5: representational state transfer (REST)," in *Architectural Styles and the Design of Network-Based Software Architectures*, University of California, Irvine, Calif, USA, 2000.
- [43] H. Cummins and T. Ward, *Enterprise OSGi in Action*, Manning, 1st edition, 2013.
- [44] Ryu, <http://osrg.github.com/ryu/>.
- [45] Project OpenDayLight Website, <http://www.opendaylight.org/project/technical-overview>.
- [46] D. Erickson, "The beacon openflow controller," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*, pp. 13–18, Hong Kong, August 2013.
- [47] Project FloodLight Website, <http://www.projectfloodlight.org/floodlight/>.
- [48] ONOS Project, <http://onosproject.org/>.
- [49] A. Karaf, <http://karaf.apache.org/>.
- [50] OpenDaylight Controller, "MD-SAL architecture," https://wiki.opendaylight.org/view/OpenDaylight_Controller:.
- [51] YANG RFC 6020, <https://tools.ietf.org/html/rfc6020>.
- [52] R. Hirannaiah, "Overview of Model-Driven SAL and Creating and Application based on MD-SAL," 2016, http://events.linuxfoundation.org/sites/events/files/slides/Radhika_Hirannaiah_MD-SAL_ONS2016.pdf.
- [53] C. Janz, *Intent NBI—Definition and Principles*, Open Networking Foundation, 2015.
- [54] M. Pham and D. B. Hoang, "SDN applications—the intent-based Northbound Interface realisation for extended applications," in *Proceedings of the IEEE NetSoft Conference and Workshops (NetSoft '16)*, pp. 372–377, Seoul, South Korea, June 2016.
- [55] ONF Blog: Intent, what. Not how, https://www.opennetworking.org/?p=1633&option=com_wordpress&Itemid=155.
- [56] T. Zhang and F. Hu, "Controller architecture and performance in software-defined networks," in *Network Innovation through OpenFlow and SDN*, CRC Press Taylor & Francis Group, Boston, Mass, USA, 2014.
- [57] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1955–1980, 2014.
- [58] ONFSummit, "What is the intent anyway," 2015, https://www.youtube.com/watch?v=QvEK_CFIGik.
- [59] Cisco, "Microservices Infrastructure," 2015, <https://github.com/CiscoCloud/microservices-infrastructure>.
- [60] ODL, "Network Information Composition," https://wiki.opendaylight.org/view/Network_Intent_Composition:Main.
- [61] ODL, Network Modelling Language (NEMO), https://wiki.opendaylight.org/view/Network_Intent_Composition:NEMO_Model.
- [62] B. J. van Asten, N. L. M. van Adrichem, and F. A. Kuipers, "Scalability and resilience of software-defined networking: an overview," <https://arxiv.org/abs/1408.6760>.
- [63] T. Koponen, M. Casado, N. Gude et al., "Onix: a distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI '10)*, pp. 351–364, Vancouver, Canada, October 2010.
- [64] A. Tootoonchian and Y. Ganjali, "Hyperflow: a distributed control plane for openflow," in *Proceedings of the Internet Network Management Conference on Research on Enterprise Networking*, p. 3, 2010.
- [65] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*, pp. 19–24, August 2012.
- [66] NOX Repository Website, <https://github.com/noxrepo/nox>.
- [67] POX, <https://openflow.stanford.edu/display/ONL/POX+Wiki>.
- [68] R. Sherwood, M. Chan, A. Covington et al., "Carving research slices out of your production networks with openflow," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 129–130, 2010.
- [69] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. Corrêa, S. C. De Lucena, and M. F. Magalhães, "Virtual routers as a service: the routeflow approach leveraging software-defined networks," in *Proceedings of the 6th International Conference on Future Internet Technologies (CFI '11)*, pp. 34–37, ACM, Seoul, Republic of Korea, June 2011.
- [70] Oflops, <http://archive.openflow.org/wk/index.php/Oflops>.
- [71] Helios by nec, <http://www.nec.com/>.
- [72] Trema openflow controller framework, <https://github.com/trema/trema>.
- [73] Jaxon:java-based openflow controller, <http://jaxon.onuos.org/>.
- [74] Mul, <http://sourceforge.net/p/mul/wiki/Home/>.
- [75] IRIS, <http://openiris.etri.re.kr/>.
- [76] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro: a system for scalable openflow control," Tech. Rep. TR10-08, Rice University, Houston, Tex, USA, December 2010.
- [77] The nodeflow openflow controller, <http://garyberger.net/?p=537>.
- [78] Network development and deployment initiative, OESS, <https://github.com/globalnoc/oess>.
- [79] SNAC Repository, <https://github.com/bigswitch/snac>.
- [80] Open vSwitch, <http://openvswitch.org/support/>.
- [81] Indigo: Open source openflow switches, <http://www.openflowhub.org/display/Indigo/>.
- [82] OpenFlowJ, <https://github.com/floodlight/loxigen/wiki/Open-FlowJ-Loxi>.
- [83] OpenFaucet, <https://github.com/rlenglet/openfaucet>.
- [84] Of softs witch13—cpqd, <https://github.com/CPqD/ofsoftswitch13>.
- [85] Pantou: Openflow 1.0 for openwrt, <http://www.openflow.org/wk/index.php/>.
- [86] Node.js, <http://nodejs.org/>.
- [87] Pica8, <http://pica8.com/products/>.
- [88] A10 Networks, https://www.a10networks.com/products/axapplication_delivery_controller.
- [89] Big vSwitch, <http://www.bigswitch.com/sites/default/files/sdn-resources/bvsdatasheet.pdf>.
- [90] Brocade ADX Series, <http://www.brocade.com/en/products-services/software-networking/application-delivery-controllers.html>.

- [91] NEC programmable switch series, <https://www.necam.com/sdn/>.
- [92] ADVA Optical—FSP 150 & 3000, <http://www.advaoptical.com/en/products/scalable-optical-transport/fsp-3000.aspx>.
- [93] IBM RackSwitch G8264, <http://www.redbooks.ibm.com/abstracts/tips0815.html>.
- [94] HP OpenFlow Enabled Switches, <https://www.hpe.com/uk/en/product-catalog/networking/networking-switches.html>.
- [95] Juniper Junos MX, EX, QFX Series, http://www.juniper.net/techpubs/en_US/junos15.1/topics/concept/virtual-chassis-ex-qfx-series-mixed-understanding.html.
- [96] Mininet, <http://mininet.org/>.
- [97] ns-3 simulator, <https://www.nsnam.org/docs/release/3.13/models/html/openflow-switch.html>.
- [98] OMNeT++, <https://omnetpp.org/>.
- [99] P. L. Ventre, B. Jakovljevic, D. Schmitz et al., “GEANT SDX-SDN based open exchange point,” in *Proceedings of the 2016 IEEE NetSoft Conference and Workshops (NetSoft '16)*, pp. 345–346, Seoul, Republic of Korea, 2016.
- [100] The OpenStack Project, <https://www.openstack.org/>.
- [101] Puppet Enterprise IT, <https://puppet.com/>.
- [102] Chef Automated Infrastructure, <https://www.chef.io/chef/>.
- [103] Critical Infrastructure Management with CFEngine, <https://cfengine.com/>.
- [104] Open Compute Project, <http://www.opencompute.org/about/>.
- [105] Facebook 6 Pack Modular Switch, <https://code.facebook.com/posts/717010588413497/introducing-6-pack-the-first-open-hardware-modular-switch/>.
- [106] Open Network Install Environment, <http://onie.org/>.
- [107] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King, “Debugging the data plane with anteater,” in *Proceedings of the ACM SIGCOMM Conference (SIGCOMM '11)*, pp. 290–301, Toronto, Canada, August 2011.
- [108] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, “VeriFlow: verifying network-wide invariants in real time,” in *Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*, pp. 49–54, ACM, Helsinki, Finland, August 2012.
- [109] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, “Where is the debugger for my software-defined network?” in *Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*, pp. 55–60, ACM, Helsinki, Finland, August 2012.
- [110] Sdn troubleshooting simulator, <http://ucb-sts.github.com/sts/>.
- [111] NICE, <https://code.google.com/archive/p/nice-of/>.
- [112] OFTest, [http://archive.openflow.org/wk/index.php/OFTestTutorial](http://archive.openflow.org/wk/index.php/OFTTestTutorial).
- [113] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann, “OFRewind: enabling record and replay troubleshooting for networks,” in *Proceedings of the USENIX Conference on USENIX Annual Technical Conference (USENIXATC '11)*, p. 29, USENIX Association, Portland, Ore, USA, June 2011.
- [114] Wireshark, <https://www.wireshark.org/>.
- [115] Edge Core Systems, <http://www.edge-core.com/productsKind.php?cls=1>.
- [116] Agema System, <http://www.agemasystems.com/products.php?ctid=20>.
- [117] iwNetworks, <http://www.iwnetworks.com/main/products/network-switches/bare-metal>.
- [118] Whitebox, Understanding the basics, <http://searchsdn.techtarget.com/tip/White-box-switches-Understanding-the-basics>.
- [119] Whiteboxes ready for prime-time. Online Article, <http://www.networkworld.com/article/3100927/network-switch/white-boxes-are-now-ready-for-prime-time.html>.
- [120] Interface Masters, <http://www.interfacemasters.com/products/switches1g-10g-40g/290>.
- [121] Alpha Networks, http://www.alphanetworks.com/en/product_detail/ef0be540e4c8723f.
- [122] Penguin Computing, <http://www.penguincomputing.com/products/network-switches/>.
- [123] BigSwitch, <http://www.bigswitch.com/products/switch-light>.
- [124] Cumulus Linux, <https://cumulusnetworks.com/cumulus-linux/overview/>.
- [125] O. S. Gigamon, <https://www.gigamon.com/products/gigavue-os>.
- [126] Broadcom, <https://www.broadcom.com/products/ethernet-communication-and-switching/switching/fastpath>.
- [127] Dell Open Networking Article, <http://www.dell.com/learn/us/en/ph/press-releases/2016-01-20-dell-raises-the-bar-for-open-networking>.
- [128] IPFusion Products, <http://www.ipinfusion.com/products/ocnos>.
- [129] M. Bansal, J. Mehlman, S. Katti, and P. Levis, “OpenRadio: a programmable wireless dataplane,” in *Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*, pp. 109–114, Helsinki, Finland, August 2012.
- [130] L. E. Li, Z. M. Mao, and J. Rexford, “Toward software-defined cellular networks,” in *Proceedings of the 1st European Workshop on Software Defined Networks (EWSN '12)*, pp. 7–12, Darmstadt, Germany, October 2012.
- [131] X. Mi, Z. Tian, X. Xu, M. Zhao, and J. Wang, “NO stack: a SDN-based framework for future cellular networks,” in *Proceedings of the International Symposium on Wireless Personal Multimedia Communications (WPMC '14)*, pp. 497–502, Sydney, Australia, September 2014.
- [132] A. Bradai, K. Singh, T. Ahmed, and T. Rasheed, “Cellular software defined networking: a framework,” *IEEE Communications Magazine*, vol. 53, no. 6, pp. 36–43, 2015.
- [133] R. H. Gau and P. K. Tsai, “SDN-based optimal traffic engineering for cellular networks with service chaining,” in *Proceedings of the 2016 IEEE Wireless Communications and Networking Conference*, pp. 1–6, Doha, Qatar, April 2016.
- [134] C. C. Marquezan, X. An, Z. Despotovic, R. Khalili, and A. Hecker, “Identifying latency factors in SDN-based Mobile Core Networks,” in *Proceedings of the IEEE Symposium on Computers and Communication (ISCC '16)*, pp. 484–491, Messina, Italy, June 2016.
- [135] R. Trivisonno, R. Guerzoni, I. Vaishnavi, and A. Frimpong, “Network resource management and QoS in SDN-enabled 5G systems,” in *Proceedings of the IEEE Global Communications Conference (GLOBECOM '15)*, pp. 1–7, San Diego, Calif, USA, 2015.
- [136] T. T. Nguyen, C. Bonnet, and J. Harri, “SDN-based distributed mobility management for 5G networks,” in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC '16)*, pp. 1–7, Doha, Qatar, April 2016.
- [137] L. M. Contreras, L. Cominardi, and H. Qian, “Mobile Networks and Applications,” *Mobile Networks and Applications*, pp. 21–226, 2016.

- [138] K.-K. Yap, R. Sherwood, M. Kobayashi et al., "Blueprint for introducing innovation into wireless mobile networks," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, pp. 25–32, New Delhi, India, September 2010.
- [139] K. K. Yap, M. Kobayashi, R. Sherwood et al., "Openroads: empowering research in mobile networks," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 125–126, 2010.
- [140] R. Saunders, J. Cho, A. Banerjee, F. Rocha, and J. Van der Merwe, "P2P offloading in mobile networks using SDN," in *Proceedings of the Symposium on SDN Research (SOSR '16)*, 7 pages, Santa Clara, Calif, USA, March 2016.
- [141] FON, <http://corp.fon.com/>.
- [142] Freifunk, <http://freifunk.net/>.
- [143] H. Huang, P. Li, S. Guo, and W. Zhuang, "Software-defined wireless mesh networks: architecture and traffic orchestration," *IEEE Network*, vol. 29, no. 4, pp. 24–30, 2015.
- [144] P. Dely, A. Kassler, and N. Bayer, "OpenFlow for wireless mesh networks," in *Proceedings of the 20th International Conference on Computer Communications and Networks (ICCCN '11)*, pp. 1–6, IEEE, August 2011.
- [145] A. Detti, C. Pisa, S. Salsano, and N. Blefari-Melazzi, "Wireless mesh software defined networks (wmSDN)," in *Proceedings of the IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob '13)*, pp. 89–95, Lyon, France, October 2013.
- [146] F. Yang, V. Gondì, J. O. Hallstrom, K.-C. Wang, and G. Eidson, "OpenFlow-based load balancing for wireless mesh infrastructure," in *Proceedings of the IEEE 11th Consumer Communications and Networking Conference (CCNC '14)*, pp. 444–449, IEEE, Las Vegas, Nev, USA, January 2014.
- [147] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, and A. Feldmann, "OpenSDWN: programmatic control over home and enterprise WiFi," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR '15)*, pp. 1–12, Santa Clara, Calif, USA, June 2015.
- [148] A. Abujoda, D. Dietrich, P. Papadimitriou, and A. Sathiseelan, "Software-defined wireless mesh networks for internet access sharing," *Computer Networks*, vol. 93, pp. 359–372, 2015.
- [149] S. Hasan, Y. Ben-David, C. Scott, E. Brewer, and S. Shenker, "Enhancing rural connectivity with software defined networks," in *Proceedings of the 3rd ACM Annual Symposium on Computing for Development (DEV '13)*, ACM, Bangalore, India, January 2013.
- [150] E. Dimogerontakis, I. Vilata, and L. Navarro, "Software defined networking for community network testbeds," in *Proceedings of the IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob '13)*, pp. 111–118, Lyon, France, October 2013.
- [151] S. Salsano, G. Siracusano, A. Detti, C. Pisa, P. L. Ventre, and N. Blefari-Melazzi, "Controller selection in a Wireless Mesh SDN under network partitioning and merging scenarios," <https://arxiv.org/abs/1406.2470>.
- [152] F. Yang, V. Gondì, J. O. Hallstrom, K.-C. Wang, and G. Eidson, "OpenFlow-based load balancing for wireless mesh infrastructure," in *Proceedings of the IEEE 11th Consumer Communications and Networking Conference (CCNC '14)*, pp. 444–449, January 2014.
- [153] J. Chung, G. Gonzalez, I. Armuelles, T. Robles, R. Alcarria, and A. Morales, "Experiences and challenges in deploying OpenFlow over real wireless mesh networks," *IEEE Latin America Transactions*, vol. 11, no. 3, pp. 955–961, 2013.
- [154] M. Seyedehbrahimi, F. Bouhafs, A. Raschellà, M. Mackay, and Q. Shi, "SDN-based channel assignment algorithm for interference management in dense Wi-Fi networks," in *Proceedings of the European Conference on Networks and Communications (EuCNC '16)*, pp. 128–132, Athens, Greece, June 2016.
- [155] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards programmable enterprise WLANs with Odin," in *Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*, pp. 115–120, ACM, Helsinki, Finland, August 2012.
- [156] R. Riggio, T. Rasheed, and F. Granelli, "EmPOWER: a testbed for network function virtualization research and experimentation," in *Proceedings of the Workshop on Software Defined Networks for Future Networks and Services (SDN4FNS '13)*, Trento, Italy, November 2013.
- [157] A. Raschellà, F. Bouhafs, M. Seyedehbrahimi, M. Mackay, and Q. Shi, "A centralized framework for smart access point selection based on the Fittingness Factor," in *Proceedings of the 23rd International Conference on Telecommunications (ICT '16)*, pp. 1–5, Thessaloniki, Greece, May 2016.
- [158] B. A. A. Nunes, M. A. S. Santos, B. T. de Oliveira, C. B. Margi, K. Obraczka, and T. Turlatti, "Software-defined-networking-enabled capacity sharing in user-centric networks," *IEEE Communications Magazine*, vol. 52, no. 9, pp. 28–36, 2014.
- [159] K. L. Huang, C. L. Liu, C. H. Gan, M. L. Wang, and C. T. Huang, "SDN-based wireless bandwidth slicing," in *Proceedings of the International Conference on Software Intelligence Technologies and Applications & International Conference on Frontiers of Internet of Things*, pp. 77–81, Hsinchu, Taiwan, 2014.
- [160] K. Nakauchi and Y. Shoji, "WiFi network virtualization to control the connectivity of a target service," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 308–319, 2015.
- [161] N. Bizanis and F. A. Kuipers, "SDN and virtualization solutions for the internet of things: a survey," *IEEE Access*, vol. 4, pp. 5591–5606, 2016.
- [162] A. De Gante, M. Aslan, and A. Matrawy, "Smart wireless sensor network management based on software-defined networking," in *Proceedings of the 27th Biennial Symposium on Communications (QBSC '14)*, pp. 71–75, Kingston, Canada, June 2014.
- [163] T. Luo, H.-P. Tan, and T. Q. S. Quek, "Sensor OpenFlow: enabling software-defined wireless sensor networks," *IEEE Communications Letters*, vol. 16, no. 11, pp. 1896–1899, 2012.
- [164] C. Orfanidis and C. Jacobsson, "Using software-defined networking principles for wireless sensor networks," in *Proceedings of the 11th Swedish National Computer Networking Workshop (SNCNW '15)*, Karlstad, Sweden, May 2015.
- [165] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "SDN-WISE: design, prototyping and experimentation of a stateful SDN solution for wireless sensor networks," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '15)*, pp. 513–521, IEEE, Hong Kong, May 2015.
- [166] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "A software defined networking architecture for the internet-of-things," in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS '14)*, pp. 1–9, IEEE, Kraków, Poland, May 2014.

- [167] P. Dely, A. Kasser, and N. Bayer, "OpenFlow for wireless mesh networks," in *Proceedings of the 20th International Conference on Computer Communications and Networks (ICCCN '11)*, pp. 1–6, August 2011.
- [168] Y. Wang, H. Chen, X. Wu, and L. Shu, "An energy-efficient SDN based sleep scheduling algorithm for WSNs," *Journal of Network and Computer Applications*, vol. 59, pp. 39–45, 2016.
- [169] A. Kumar, S. Jain, U. Naik et al., "BwE: flexible, hierarchical bandwidth allocation for WAN distributed computing," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*, pp. 1–14, London, UK, August 2015.
- [170] P. Patel, D. Bansal, L. Yuan et al., "Ananta: cloud scale load balancing," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '13)*, pp. 207–218, August 2013.
- [171] S. Natarajan, A. Ramaiah, and M. Mathen, "A software defined cloudgateway automation system using OpenFlow," in *Proceedings of the IEEE 2nd International Conference on Cloud Networking (CloudNet '13)*, pp. 219–226, San Francisco, Calif, USA, November 2013.
- [172] B. Heller, S. Seetharaman, P. Mahadevan et al., "ElasticTree: saving energy in data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, p. 17, USENIX Association, 2010.
- [173] N. Beheshti and Y. Zhang, "Fast failover for control traffic in software-defined networks," in *Proceedings of the IEEE Global Communications Conference (GlobeCom '12)*, pp. 2689–2694, Ericsson Research, Anaheim, Calif, USA, 2012.
- [174] J. Metzler, "Understanding software-defined networks," InformationWeek Reports, 2012, <http://reports.informationweek.com/abstract/6/9044/Data-Center/research-understanding-software-defined-networks.html>.
- [175] C. D. Marsan, "IAB Panel Debates Management Benefits, Security Challenges of Software-Defined Networking," IETF Journal, October 2012.
- [176] V. Gudla, S. Das, A. Shastri et al., "Experimental demonstration of openflow control of packet and circuit switches," in *Proceedings of the Optical Fiber Communication Conference (OFC '11), Collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC '10)*, vol. 45, pp. 1–3, IEEE, Los Angeles, Calif, USA, 2011.
- [177] D. Simeonidou, R. Nejabati, and M. P. Channegowda, "Software defined optical networks technology and infrastructure: enabling software-defined optical network operations," in *Proceedings of the Optical Fiber Communication Conference (OFC '13)*, Optical Society of America, March 2013.
- [178] L. Liu, T. Tsuritani, I. Morita, H. Guo, and J. Wu, "OpenFlow-based wavelength path control in transparent optical networks: a proof-of-concept demonstration," in *Proceedings of the 37th European Conference on Optical Communication and Exhibition (ECOC '11)*, pp. 1–3, September 2011.
- [179] A. N. Patel, P. N. Ji, and T. Wang, "Qos-aware optical burst switching in openflow based software-defined optical networks," in *Proceedings of the 17th International Conference on Optical Network Design and Modeling (ONDM '13)*, pp. 275–280, Brest, France, April 2013.
- [180] "Optical transport working group otwg. In Open Networking Foundation ONE," 2013.
- [181] K. L. Calvert, W. K. Edwards, N. Feamster, R. E. Grinter, Y. Deng, and X. Zhou, "Instrumenting home networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 84–89, 2011.
- [182] N. Feamster, "Outsourcing home network security," in *Proceedings of the ACM SIGCOMM Workshop on Home Networks (HomeNets '10)*, pp. 37–42, New Delhi, India, September 2010.
- [183] R. Mortier, T. Rodden, T. Lodge et al., "Control and understanding: owning your home network," in *Proceedings of the 4th International Conference on Communication Systems and Networks (COMSNETS '12)*, pp. 1–10, IEEE, Bangalore, India, January 2012.
- [184] M. Dillon and T. Winters, "Virtualization of home network gateways," *Computer*, vol. 47, no. 11, Article ID 6965269, pp. 62–65, 2014.
- [185] J. Jo, S. Lee, and J. W. Kim, "Software-defined home networking devices for multi-home visual sharing," *IEEE Transactions on Consumer Electronics*, vol. 60, no. 3, pp. 534–539, 2014.
- [186] A. Takacs, E. Bellagamba, and J. A. Wilke, "Software-defined networking: the service provider perspective," *Ericsson Review*, February 2013.
- [187] S. Mehdi, J. Khalid, and S. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *Recent Advances in Intrusion Detection*, pp. 161–180, Springer, Berlin, Germany, 2011.
- [188] Z. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt, and G. Noubir, "Application awareness in SDN," *SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 487–488, 2013.
- [189] H. Mekky, F. Hao, S. Mukherjee, Z.-L. Zhang, and T. V. Lakshman, "Application-aware data plane processing in SDN," in *Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking (HotSDN '14)*, pp. 13–18, ACM, New York, NY, USA, 2014.
- [190] H. E. Egilmez and A. M. Tekalp, "Distributed QoS architectures for multimedia streaming over software defined networks," *IEEE Transactions on Multimedia*, vol. 16, no. 6, pp. 1597–1609, 2014.
- [191] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, "SDN-based application-aware networking on the example of youtube video streaming," in *Proceedings of the 2nd European Workshop on Software Defined Networks (EWSDN '13)*, pp. 87–92, Berlin, Germany, October 2013.
- [192] J. Ruckert, J. Blendin, and D. Hausheer, "RASP: using OpenFlow to push overlay streams into the Underlay," in *Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing (P2P '13)*, Trento, Italy, September 2013.
- [193] C. A. C. Marcondes, T. P. C. Santos, A. P. Godoy, C. C. Viel, and C. A. C. Teixeira, "CastFlow: clean-slate multicast approach using in-advance path processing in programmable networks," in *Proceedings of the 17th IEEE Symposium on Computers and Communication (ISCC '12)*, pp. 000094–000101, IEEE, Cappadocia, Turkey, July 2012.
- [194] K. A. Noghani and M. O. Sunay, "Streaming multicast video over software-defined networks," in *Proceedings of the 11th IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS '14)*, pp. 551–556, Philadelphia, Pa, USA, October 2014.
- [195] P. Panwaree, K. Jongwon, and C. Aswakul, "Packet delay and loss performance of streaming video over emulated and real openflow networks," in *Proceedings of the 29th International Technical Conference on Circuit/Systems Computers and Communications (ITC-CSCC '14)*, At Phuket, Thailand, 2014.
- [196] Microsoft Lync SDN API, <https://msdn.microsoft.com/en-us/library/office/dn387069.aspx>.

- [197] G. Liu and T. Wood, "Cloud-scale application performance monitoring with SDN and NFV," in *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E '15)*, pp. 440–445, March 2015.
- [198] V. Mann, A. Vishnoi, and S. Bidkar, "Living on the edge: monitoring network flows at the edge in cloud data centers," in *Proceedings of the 5th International Conference on Communication Systems and Networks (COMSNETS '13)*, pp. 1–9, IEEE, Bangalore, India, January 2013.
- [199] NetFlow, <http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html>.
- [200] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: high performance and flexible networking using virtualization on commodity platforms," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 34–47, 2015.
- [201] X. N. Nguyen, D. Saucez, and T. Turletti, "Efficient caching in content-centric networks using openflow," in *Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM '13)*, pp. 67–68, Rome, Italy, April 2013.
- [202] L. Veltri, G. Morabito, S. Salsano, N. Blefari-Melazzi, and A. Detti, "Supporting information-centric functionality in software defined networks," in *Proceedings of the IEEE ICC Workshop on Software Defined Networks*, June 2012.
- [203] N. B. Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, and L. Veltri, "An OpenFlow-based testbed for information centric networking," in *Proceedings of the 21st Future Network & Mobile Summit (FutureNetw '12)*, pp. 1–9, IEEE, Berlin, Germany, July 2012.
- [204] J. Suh, H. Jung, T. Kwon, and Y. Choi, "C-flow: content-oriented networking over openflow," in *Proceedings of the Open Networking Summit (ONS '12)*, Santa Clara, Calif, USA, April 2012.
- [205] D. Syrivelis, G. Parisi, D. Trossen et al., "Pursuing a software defined information-centric network," in *Proceedings of the 1st European Workshop on Software Defined Networks (EWSDN '12)*, pp. 103–108, October 2012.
- [206] X. N. Nguyen, *Software defined networking in wireless mesh network [M.S. thesis]*, INRIA, University of Nice Sophia Antipolis, Nice, France, 2012.
- [207] "Cisco visual networking index: global mobile data traffic forecast update, 2011–2016," Tech. Rep., Cisco, 2012.
- [208] R. N. B. Rais, M. Mendonca, T. Turletti, and K. Obraczka, "Towards truly heterogeneous internets: bridging infrastructure-based and infrastructure-less networks," in *Proceedings of the 3rd International Conference on Communication Systems and Networks (COMSNETS '11)*, pp. 1–10, Bangalore, India, January 2011.
- [209] A. Coyle and H. Nguyen, "A frequency control algorithm for a mobile adhoc network," in *Proceedings of the Military Communications and Information Systems Conference (MilCIS '10)*, Canberra, Australia, November 2010.
- [210] Y. Zhang, N. Beheshti, and M. Tatipamula, "On resilience of split-architecture networks," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '11)*, pp. 1–6, 2011.
- [211] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *420 Acm Sigcomm Computer Communication Review*, vol. 42, no. 4, pp. 7–12, 2012.
- [212] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 19, no. 1, pp. 30–33, 2015.
- [213] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 18, no. 8, pp. 1339–1342, 2014.
- [214] M. F. Bari, A. R. Roy, S. R. Chowdhury et al., "Dynamic controller provisioning in software defined networks," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM '13)*, pp. 18–25, Zürich, Switzerland, October 2013.
- [215] F. A. Özsoy and M. Ç. Pinar, "An exact algorithm for the capacitated vertex p-center problem," *Computers & Operations Research*, vol. 33, no. 5, pp. 1420–1436, 2006.
- [216] L. Yao, P. Hong, W. Zhang, J. Li, and D. Ni, "Controller placement and flow based dynamic management problem towards SDN," in *Proceedings of the IEEE International Conference on Communication Workshop (ICCW '15)*, pp. 363–368, IEEE, London, UK, June 2015.
- [217] F. J. Ros and P. M. Ruiz, "On reliable controller placements in Software-Defined Networks," *Computer Communications*, vol. 77, pp. 41–51, 2016.
- [218] T. Erlebach, A. Hall, L. Moonen, A. Panconesi, F. Spieksma, and D. Vukadinovi, "Robustness of the internet at the topology and routing level," *Access and Download Statistics*, vol. 4028, pp. 260–274, 2006.
- [219] M. Guo and P. Bhattacharya, "Controller placement for improving resilience of software-defined networks," in *Proceedings of the 4th IEEE International Conference on Networking and Distributed Computing (ICNDC '13)*, pp. 23–27, IEEE, Los Angeles, Calif, USA, December 2013.
- [220] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Physical Review E Statistical Nonlinear and Soft Matter Physics*, vol. 70, no. 6, pp. 264–277, 2004.
- [221] S. Guo, S. Yang, Q. Li, and Y. Jiang, "Towards controller placement for robust software-defined networks," in *Proceedings of the IEEE 34th International Performance Computing and Communications Conference (IPCCC '15)*, pp. 1–8, Nanjing, China, December 2015.
- [222] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "Reliability-aware controller placement for software-defined networks," *Wireless Communication Over Zigbee for Automotive Inclination Measurement China Communications*, vol. 11, no. 2, pp. 672–675, 2013.
- [223] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," in *Proceedings of the ACM SIGCOMM 2010 conference (SIGCOMM '10)*, pp. 351–362, New Delhi, India, September 2010.
- [224] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 254–265, 2011.
- [225] A. Voellmy, H. Kim, and N. Feamster, "ProCera: a language for high-level reactive network control," in *Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*, pp. 43–48, ACM, Helsinki, Finland, August 2012.
- [226] N. Foster, R. Harrison, M. J. Freedman et al., "Frenetic: a network programming language," in *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP '11)*, pp. 279–291, ACM, Tokyo, Japan, September 2011.
- [227] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Practical declarative network management," in *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking (WREN '09)*, pp. 1–10, New York, NY, USA, 2009.

- [228] A. Voellmy and P. Hudak, "Nettle: taking the sting out of programming network routers," in *Practical Aspects of Declarative Languages: 13th International Symposium, PADL 2011, Austin, TX, USA, January 24-25, 2011. Proceedings*, vol. 6539 of *Lecture Notes in Computer Science*, pp. 235–249, Springer, Berlin, Germany, 2011.
- [229] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software-defined networks," in *Proceedings 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13)*, Lombard, Ill, USA, 2013.
- [230] Y. Luo, P. Cascon, E. Murray, and J. Ortega, "Accelerating OpenFlow switching with network processors," in *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '09)*, pp. 70–71, ACM, Princeton, NJ, USA, October 2009.
- [231] A. Nakao, "Flare: open deeply programmable network node architecture," http://netseminar.stanford.edu/seminars/10_18_12.pdf.
- [232] Controller performance comparisons, <http://www.openflow.org/wk/index.php/>.
- [233] K. Benton, L. J. Camp, and C. Small, "OpenFlow vulnerability assessment," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*, pp. 151–152, Hong Kong, August 2013.
- [234] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*, pp. 1–6, ACM, Helsinki, Finland, August 2012.
- [235] J. Metzler, "Understanding Software-Defined Networks," InformationWeek Reports, pp.1–25, October 2012, <http://reports.informationweek.com/abstract/6/9044/Data-Center/research-understanding-software-defined-networks.html>.
- [236] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2016.
- [237] S. Shin and G. Gu, "Attacking software-defined networks: a first feasibility study," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*, pp. 165–166, August 2013.
- [238] R. Smeliansky, "SDN for network security," in *Proceedings of the International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC '14)*, pp. 1–5, Moscow, Russia, October 2014.
- [239] L. Schehlmann, S. Abt, and H. Baier, "Blessing or curse? Revisiting security aspects of software-defined networking," in *Proceedings of the 10th International Conference on Network and Service Management (CNSM '14)*, pp. 382–387, Rio de Janeiro, Brazil, November 2014.
- [240] A. Y. Ding, J. Crowcroft, S. Tarkoma, and H. Flinck, "Software defined networking for security enhancement in wireless mobile networks," *Computer Networks*, vol. 66, pp. 94–101, 2014.
- [241] D. Kreutz, F. M. V. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*, pp. 55–60, Hong Kong, August 2013.
- [242] M. Tsugawa, A. Matsunaga, and J. A. Fortes, "Cloud computing security: what changes with software-defined networking?" in *Secure Cloud Computing*, pp. 77–93, Springer, New York, NY, USA, 2014.
- [243] OpenFlowSec, <http://www.openflowsec.org/>.
- [244] J. Hizver, "Taxonomic modeling of security threats in software defined networking," in *Proceedings of the BlackHat Conference*, August 2015.
- [245] Border Gateway Protocol (BGP), <https://tools.ietf.org/html/rfc4271>.
- [246] Open Shortest Path First (OSPF), <https://tools.ietf.org/html/rfc2328>.
- [247] A. Gämperli, V. Kotronis, and X. Dimitropoulos, "Evaluating the effect of centralization on routing convergence on a hybrid BGP-SDN emulation framework," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 369–370, 2014.
- [248] W. Duan, L. Xiao, D. Li et al., "OFBGP: a scalable, highly available BGP architecture for SDN," in *Proceedings of the 11th IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS '14)*, pp. 557–562, IEEE, Philadelphia, Pa, USA, October 2014.
- [249] P. Lin, J. Bi, and H. Hu, "BTSDN: BGP-based transition for the existing networks to SDN," in *Proceedings of the 6th International Conference on Ubiquitous and Future Networks (ICUFN '14)*, pp. 419–424, Shanghai, China, July 2014.
- [250] C. Chen, B. Li, and D. Lin, "Software-defined inter-domain routing revisited," in *Proceedings of the IEEE International Conference on Communications (ICC '16)*, pp. 1–6, Kuala Lumpur, Malaysia, May 2016.
- [251] L. You, L. Wei, L. Junzhou, J. Jian, and X. Nu, "An inter-domain multi-path flow transfer mechanism based on SDN and multi-domain collaboration," in *Proceedings of the 14th IFIP/IEEE International Symposium on Integrated Network Management (IM '15)*, pp. 758–761, Ottawa, Canada, May 2015.
- [252] M. Caria, T. Das, A. Jukan, and M. Hoffmann, "Divide and conquer: partitioning OSPF networks with SDN," in *Proceedings of the 14th IFIP/IEEE International Symposium on Integrated Network Management (IM '15)*, pp. 467–474, Ottawa, Canada, May 2015.
- [253] Z. Chen, J. Bi, Y. Fu, Y. Wang, and A. Xu, "MLV: a multi-dimension routing information exchange mechanism for inter-domain SDN," in *Proceedings of the IEEE 23rd International Conference on Network Protocols (ICNP '15)*, pp. 438–445, San Francisco, Calif, USA, November 2015.
- [254] Y. Wang, J. Bi, K. Zhang, and Y. Wu, "A framework for fine-grained inter-domain routing diversity via SDN," in *Proceedings of the 8th International Conference on Ubiquitous and Future Networks (ICUFN '16)*, pp. 751–756, Vienna, Austria, July 2016.
- [255] K. Phemius, M. Bouet, and J. Leguay, "DISCO: distributed SDN controllers in a multi-domain environment," in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS '14)*, pp. 1–2, IEEE, Kraków, Poland, May 2014.
- [256] P. Thai and J. C. de Oliveira, "Decoupling policy from routing with software defined interdomain management: interdomain routing for SDN-based networks," in *Proceedings of the IEEE 22nd International Conference on Computer Communication and Networks (ICCCN '13)*, pp. 1–6, Nassau, Bahamas, August 2013.
- [257] Quagga Routing Suite, <http://www.nongnu.org/quagga/>.
- [258] A. Gupta, L. Vanbever, M. Shahbaz et al., "SDX, a software defined internet exchange," in *Proceedings of the ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 551–562, ACM, Chicago, Ill, USA, August 2014.

- [259] S. Whyte, Project CARDIGAN. An SDN Controlled Exchange Fabric, 2012, <https://www.nanog.org/meetings/nanog57/presentations/Wednesday/wed.lightning3.whyte.sdn.controlled.exchange.fabric.pdf>.
- [260] “OpenDay Light Software-Defined Network Interfac Application (SDNi),” https://wiki.opendaylight.org/view/ODL-SDNi_App:Main.
- [261] “SDN-Enabled Virtual SBC/Virtual IMS,” <http://h20195.www2.hp.com/V2/GetPDF.aspx/4AA6-3418ENW.pdf>.
- [262] Inter-SDN controller communication, <http://www.tcs.com/SiteCollectionDocuments/White%20Papers/Inter-SDN-Controller-Communication-Border-Gateway-Protocol-0314-1.pdf>.

